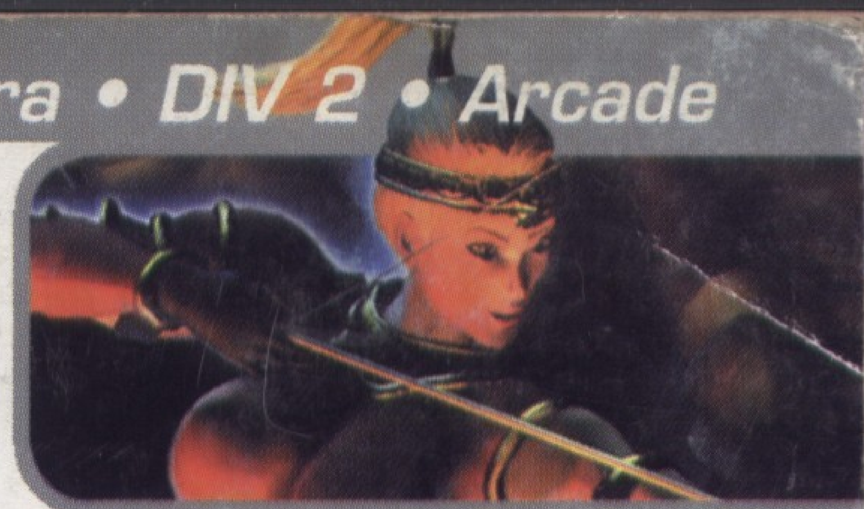
Premios para los lectores • Estrategia • C y Ensamblador • Aventura • DIV 2 • Arcade





CONCURSO JUEGOS DIV

DIV manía

995 ptas.

PORTUGAL 990 ESC (CONT) 5,98 €

www.prensatecnica.com

Año 2 • Número 6

• DIV 2
Manejando cadenas de texto

• 3D
Creando un juego: DIV Deathmaker

• DIV interno
Aprendemos a utilizar archivos de mapas

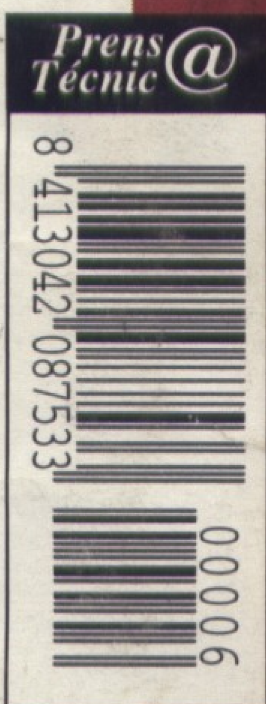
• Webs DIV
VitalWeb, otra revista electrónica sobre DIV

• Programación
Aprende a hacer juegos de estrategia, rol y aventuras

• Div Iniciación
Empieza a programar juegos ahora mismo

• Utilidades
FreeMem Pro, liberador de memoria de Windows

• Share Música
Crea la banda sonora de tus juegos



Inteligencia Artificial

¿Sueñan los androides con ovejas eléctricas?



en el CD-Rom **50 JUEGOS Div**

DIVmanía



¿Amas la programación de juegos...? Esta es tu revista

Parece que se ha creado cierta polémica respecto a nuestra revista y su vinculación con Hammer Technologies y nos gustaría aclarar un par de cuestiones que nos parecen importantes.

Es cierto que Hammer Technologies y Prensa Técnica, editora de nuestra revista, pertenecen al mismo grupo; sin embargo son dos empresas independientes. Es posible que, en un principio, la revista dependiera, de un modo u otro, de Hammer pero, en la actualidad y, al menos, desde que está el actual equipo, no tenemos relación de dependencia alguna.

La nuestra no es una revista dedicada a la promoción de Hammer, y la redacción de Divmanía trata, simplemente, de hacer una revista independiente y objetiva sobre DIV. Por este hecho, no tenemos nada que ver con la marcha de gente de Hammer o con los planes de ésta (planes que, por cierto, desconocemos) y mucho menos con su política.

Sabemos que nuestra revista no es, ni mucho menos, perfecta, pero nos esforzamos por aprender e intentamos hacer que sea útil y amena. Nos agrada enormemente recibir cartas de nuestros lectores en las que nos felicitan. Por supuesto también llegan críticas pero, afortunadamente, son menos que las felicitaciones. Esto no es óbice para que las despreciemos; al contrario: tratamos de tomarlas como un acicate para seguir mejorando. Y en eso estamos.

Ponte al día

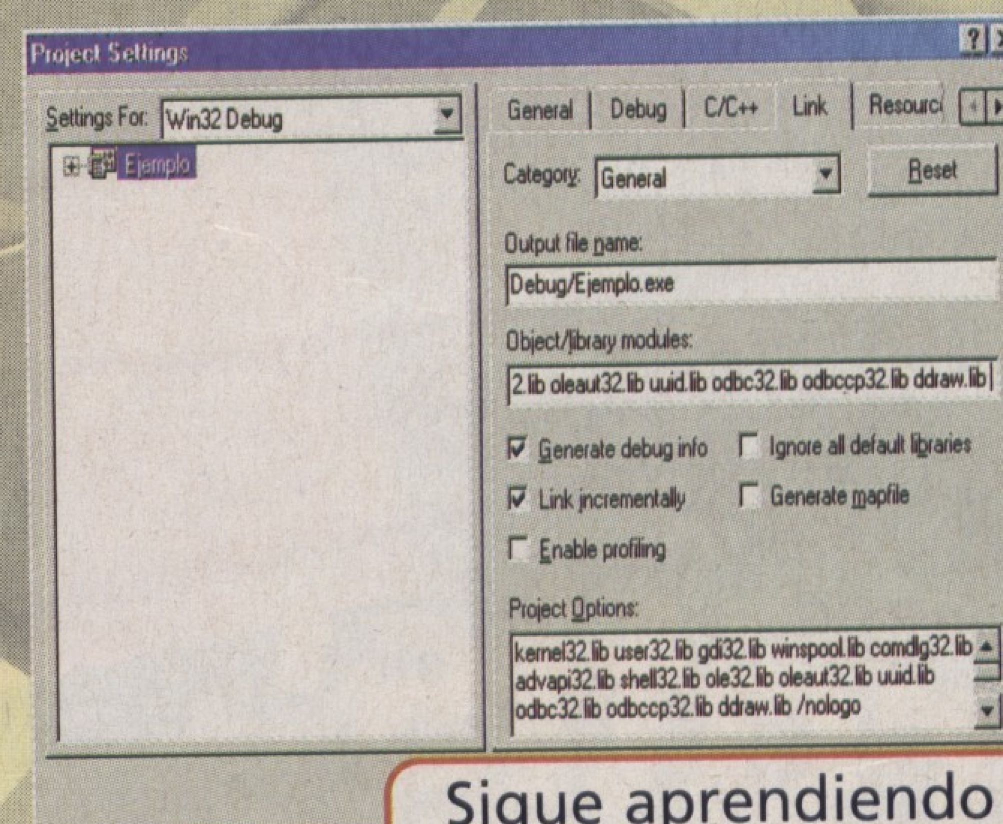


DIV en la red



DIV Developer

Sigue aprendiendo



Director: Mario Luis
mluis@prensatecnica.com
Coordinador Técnico: Oscar Condés
gover@prensatecnica.com
Colaboradores: Pablo Trinidad, Sergio Cánovas, Miguel Barroso, David Martínez, Emilio Llamas, Javier Fernández, Fermín Vicente, Armando Vélez, Ramón de España y Daniel García.
Edición: Alfredo del Barrio, Guillermo Izquierdo y Martín Moncalvillo

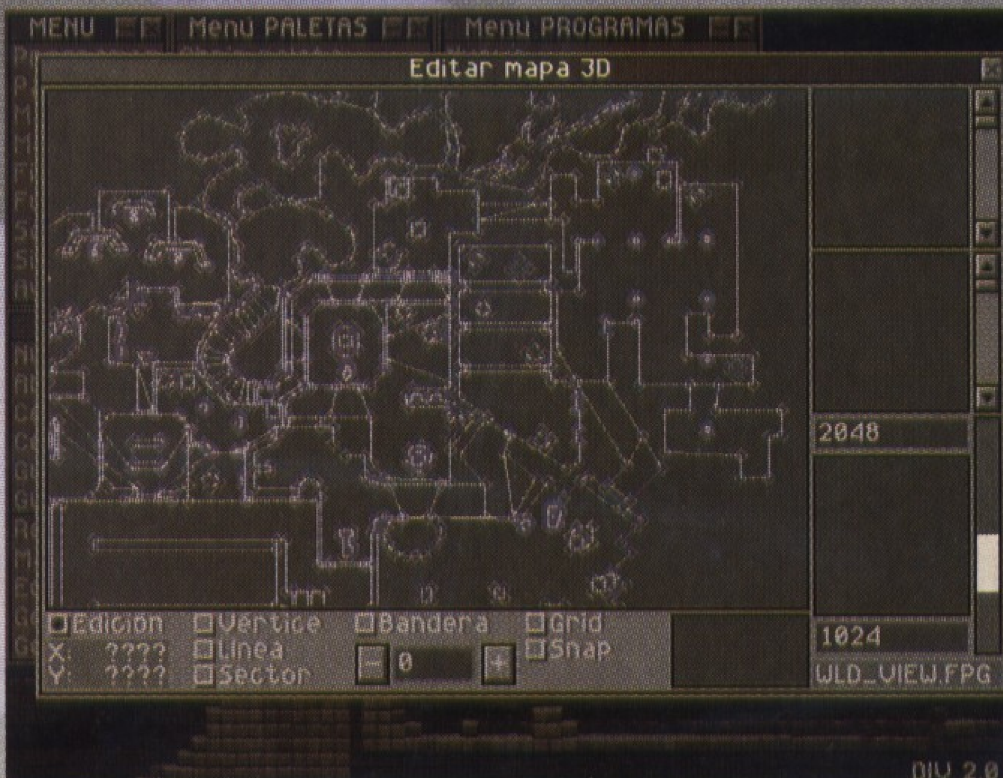
Dirección de Arte: Francisco Calero
Jefe de Departamento: José A. Gil
Maquetación: Antonio García Tomé, Antonio Barbero, Ana Isabel Madero, Susana Burgaleta, Jesús Mena, Oscar Menoyo y Mercedes Albizua
Portada: Francisco A. Anguís
Publicidad: Marisa Fernández, marisa@prensatecnica.com
Sonia Glez.-Villamil, Noelia Menéndez y Emerio Arena

Supervisión CD-Rom: Alvaro García
Servicio Técnico CD-Rom: Eugenio García
Horario de atención: tardes 16:00 - 18:00 h
E-mail: stecnico@prensatecnica.com
Secretaría de Redacción: Elena Fernández
Departamento de Suscripciones: Sandra Fernández y Noemí Iscart
suscripciones@prensatecnica.com

Departamento de Administración: Juan López, Juan Ignacio Domínguez
Departamento Comercial: Marcelino Ormeño
REDACCIÓN, PUBLICIDAD Y ADMINISTRACIÓN
c/ Alfonso Gómez 42. Nave 1.1.2
Madrid 28037. España
Tfno: 91. 304. 06. 22
Fax: 91. 304. 17. 97
Si llama desde fuera de España, marcar (+34)

Sumario





Programas del lector



ESPECIAL JUEGOS DIV

Muchos de nuestros lectores nos habían hecho llegar un mensaje muy claro: publicadlos, aunque no los premiéis. Pues bien, ahí los tenéis. Os ofrecemos más de 50 juegos para que disfrutéis y aprendáis.

¿QUÉ ES DIV GAMES STUDIO?

DIV Games Studio es una herramienta de programación que facilita en gran manera nuestra inmersión en el software de entretenimiento. Es el primer entorno profesional que permite realizar videojuegos con fines comerciales sin necesidad de un pago adicional. Con el carné de desarrollador incluido se permite el desarrollo de cualquier tipo de juegos y su libre venta y distribución.



E-mail: gover@prensatecnica.com
http://www.prensatecnica.com
Horario de atención al público: de 09:00 a 19:00 h ininterrumpidamente

EDITA: PRENSA TÉCNICA
Director General: Mario Luis
Director Editorial: Eduardo Toribio
Director Técnico: Fernando Escudero
Director de Producción: C.P. Cerezo
Directora Publicidad: Marisa Fernández
Director Comercial: Esteban Martinez
Fotomecánica: Prensa Técnica
Impresión: Pantone, S.A.
Duplicación del CD-Rom: M.P. O., Servicios Ibéricos, Grupo Cóndor

Distribución:
SGEL. Avda Valdelaparra, 29
Alcobendas. Madrid

Depósito legal: M-42077-1998
AÑO 2 • NÚMERO 6

El lector lúdico

Sólo para aficionados a los juegos



El equipo de redacción de Divmanía.

Hemos comprobado fehacientemente que nuestra revista está dando mucho que hablar. Para bien, y de eso nos alegramos enormemente, y para mal, lo que consideramos como algo bueno ya que las críticas deben ser aceptadas por el lado positivo, pensando en lo que nos ayudan a superarnos a nosotros mismos. Y en eso estamos.

En primer lugar recogemos una duda que surge en los últimos tiempos alrededor de Div y su futuro. Recogemos un extracto de algunos mensajes que nos han enviado nuestros lectores:

(...) he leído en una revista de informática, que el nuevo Windows 2000 no soporta los programas en MS-DOS, en una palabra, quieren jubilar este sistema. ¿Qué pasará con Div2? ¿Cuándo Div2 saltara a la plataforma Windows? Ya sé que esto no es fácil, pero estamos ansiosos de que Hammer tome esta decisión y poder realizar aplicaciones Windows con Div. Creo que merece la pena que Div siga adelante y no se quede como una aventura. ¡¡Hammer, no nos dejes en la estacada, muchos confiamos en Div, puede tener mucha y larga vida si se hace este salto!!

(...) Y aunque esto sé que no se puede cumplir... que se publique los planes actuales de Hammer para DiV, o la plantilla que confoma div ahora (0?), yo que sé, algo que no nos deje en vilo...

Pues bien, en referencia a este tema, simplemente recordaros lo que ya apuntamos en nuestra editorial: sencillamente que Divmanía es una revista totalmente independiente de Hammer Technologies. Sin embargo, nos hemos dirigido a ellos para consultarles estas dudas y su respuesta ha sido la siguiente: *Los planes de Hammer todavía no se pueden desvelar, pero habrá gratas sorpresas para sus más fieles seguidores. DIV para Windows es uno de los proyectos en fase de estudio en Hammer para este año.*

Sentimos no poder deciros nada más pero, sencillamente, tal y como os hemos comentado, Divmanía no tiene ninguna relación de dependencia con Hammer. La prueba de que no somos una revista mediatizada está en que no hemos tenido reparos en publicar las críticas que nos llegan:

Ampliación a más páginas de la revista... inclusión de la caja para el cd, por lo menos un cartón... algo que demuestre un gesto de voluntad de los de Hammer joroña...

(...) llevo comprando la revista 3 números, y cada vez me repito que es la última, el contenido, quitando algunos artículos, me parece una reutilización continua de artículos entre todas las revistas del grupo (...)

DiV developer parecía al principio de la publicación de DiVmanía una sección prometedora que iría encaminada a enseñar con el ejemplo. Y en parte lo es porque en lo que respecta a las secciónes de Programación Básica, C y Ensamblador no hay nada importante que comentar. No obstante en lo referente al comentario de los Juegos Ganadores se limitan a transcribir lo que su autor puso sobre el juego y parte del código. Esto último me parece poco profesional pues se debería comentar aquellas partes del código que (...) destaquen sobre el resto y debe ser tenido en cuenta por los usuarios (...).

Las noticias por lo general carecen de todo interés por parte de los usuarios DiV. Deben tener en cuenta que DiVmanía es la única revista que está especializada en el producto de programación de Hammer Technologies. Por tanto es aquí donde se esperan encontrar cosas que el usuario desconozca y que tengan relación realmente con su programación o uso. (...) Otro aspecto que en DiVmanía se echa en falta es la actualidad... En mi opinión deberían contratar a alguna persona que estuviera cerca del mundo DiV día a día, que se preocupe por conocer las noticias que importan... que se entere de las kedadas que se producen en las distintas ciudades de España...(...), que sepa cuáles son las páginas más actualizadas, las últimas noticias sobre los proyectos... ésas son las verdaderas noticias que interesan...

Por lo general los reportajes suelen entrar poco en detalle quedándose, en muchas ocasiones, simplemente en lo superficial... si hiciera falta creo que sería conveniente hacer reportajes por entregas, porque cuando un tema interesa sobremanera resulta insaciable que se trate en tan solo un par de páginas. Los usuarios un tanto más aventajados han conseguido llegar al nivel que se describe en los reportajes de forma autodidácta y desea profundizar dentro de DiV en esos mismos aspectos, algo que no le proporciona DiVmanía.

El contenido de los CD's que acompañan con la revista, exceptuando las utilidades más generales y necesarias (winzip, getright, mirc, acdsee...), no suelen tener ningún tipo de interés y da la impresión (por no afirmalo de todo punto) de ser producto de bulto. (...) No es adecuado ni sirve de nada ofrecer programas que comprimen documentos de texto en una sola página para su impresión o programas que hacen patches de actualizaciones... ¿es que se inserta lo primero que se ve por internet? A mi modesto entender, el problema de DiVmanía se basa en que realmente la publicación no centra sus objetivos en DiV, se denota una carencia de material y de personal realmente especializado. Y creo hablar en nombre de muchos (...)



Pues bien, simplemente decir que tomamos nota de las críticas y nos ponemos a trabajar para mejorar nuestra revista. Muchas gracias a todos.

Autónomos, empresarios, nóminas, cuentas, facturas

Por sólo 9.950 pts. podrá resolver la contabilidad de su empresa o actividad

PCautónomos

Un programa que le aporta todas las soluciones:

- Adaptado al **Euro.**
- **Preparado** para el **2000.**
- **Licencia** y **garantía.**
- **Facilidad** de instalación.
- **Mínimos** requisitos informáticos.
- Soluciones de gestión **económico-financiera.**
- **Modelos** tributarios **oficiales.**
- **Facturación**, **Libros**, **Ficheros**, **Cuentas**, **Extractos**, **Recibos**: todas las aplicaciones que necesita.
- **99** Titulares Profesionales.
- **Botones** de acceso rápido.
- **Ayuda** permanente en **pantalla.**
- **Manual** en **línea.**
- **Multiejercicios.**



• Programa PC autónomos
+ Manual de usuario
+ Guía del autónomo

NOVEDAD

Software para la gestion en PC de

Contabilidad + Nominas

Solucione todos los problemas de Nominas, Contabilidad y Facturacion de su empresa con facilidad

Programas NominaPro ContaPro FactuPro

Solo **9.950** pta

59,80 Euros

3 Programas completos

Más de 150.000 usuarios

• Solucione su proceso de adaptación técnica al año 2000

• Tres programas informáticos que le permitirán resolver todas las necesidades operativas de su empresa

ContaPro euro 2000
NominaPro euro 2000
FactuPro euro 2000

Conta Pro euro 2000
- Balances, Cuentas, gestión de Cobros y Pagos, Bibliotecas, Asientos: cubre todas las necesidades de contabilidad.
- Número ilimitado de apuntes y empresas.

Nomina Pro euro 2000
- Archivo, Procesos, Listados, I.R.P.F., Útiles, Opciones y Ayuda.
- Incidencias, Nóminas, Seguros Sociales.

Factu Pro euro 2000
- Albaranes, Facturas, Facturas de Proveedor, Almacén, Estadísticas y Gráficas, Utilidades.
- Existencias, Stock, Entrega y Medio de Envío.

PRENSA TÉCNICA • C/ Alfonso Gómez nº 42, nave 1-1-2. 28037 Madrid • Fax: 91 304 17 97 • www.prensatecnica.com

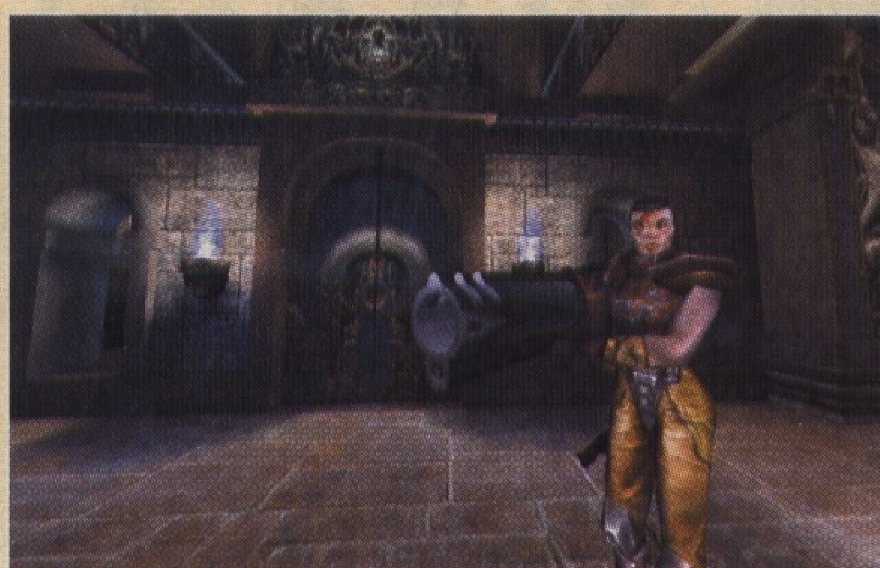
SOLICITE INFORMACIÓN EN EL TELÉFONO 91 304 06 22

Quake TV

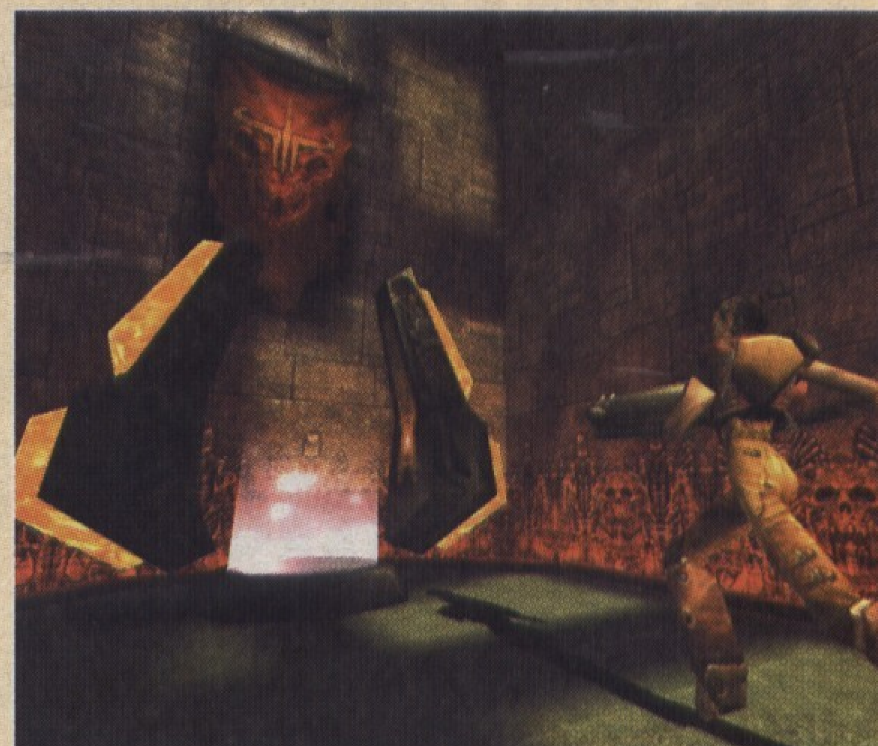


La epidemia que atraviesa el mundo de punta a punta se llama *Quake*. Se caracteriza por convulsiones en la muñeca, ojos saltones y enrojecidos, cara de loco peligroso y dificultades para pagar la factura del teléfono a final de mes.

Lo último de lo último, ideal para las pequeñas pausas obligatorias para meter la muñeca en hielo, es ver partidas que se estén desarrollando en Internet sin participar en ellas, como si de una película de acción se tratase.



Quake TV es una página de Internet que permitirá ver en directo partidas de *Quake II* y *Quake III* a través del ordenador. Se podrán elegir entre tres modalidades de visión dependiendo del programa, totalmente gratuito, que descarges:



RocCam, a cámara parada; RocRelay, se verá como si realmente estuviésemos participando en el juego a través de una cámara situada en la cabeza de un participante; y RocStream, para las mejores escenas.
Podréis encontrar más información en la siguiente dirección:
http://savageuk.com/quaketv/



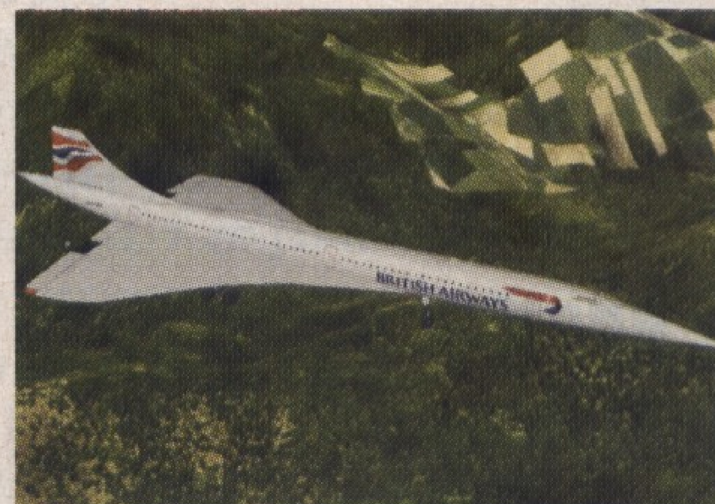
Los más vendidos en EEUU

La agencia PCDATA ha informado de cuál es la lista de los juegos más vendidos en Estados Unidos durante diciembre del año pasado. Este hit-parade del software lúdico es el siguiente:

1. Who wants to be a millionaire, de Disney.
2. MP Roller Coaster Tycoon, de Hasbro Interactive.
3. Barbie Generation Girl Gotta Groove, de Mattel Interactive.
4. Age Of Empires II: Age of Kings, de Microsoft.
5. Quake III Arena, de Id.
6. Toy Story 2 Action Game, de Disney.
7. Deer Hunter III, de GT Interactive.
8. Frogger, de Hasbro Interactive.
9. Cabela´s Big Game Hunter 3, de Activision.
10. Microsoft Flight Simulator, de Microsoft.







Como podéis observar, las peculiaridades del país americano hacen que los primeros puestos de la lista estén ocupados por títulos dirigidos eminentemente a niños, el primer puesto es un juego de Disney basado en un concurso televisivo que no creemos que vea la luz en nuestro país. También sorprende que entre los diez primeros juegos estén representados dos títulos de simulación de caza. En fin, ellos verán.

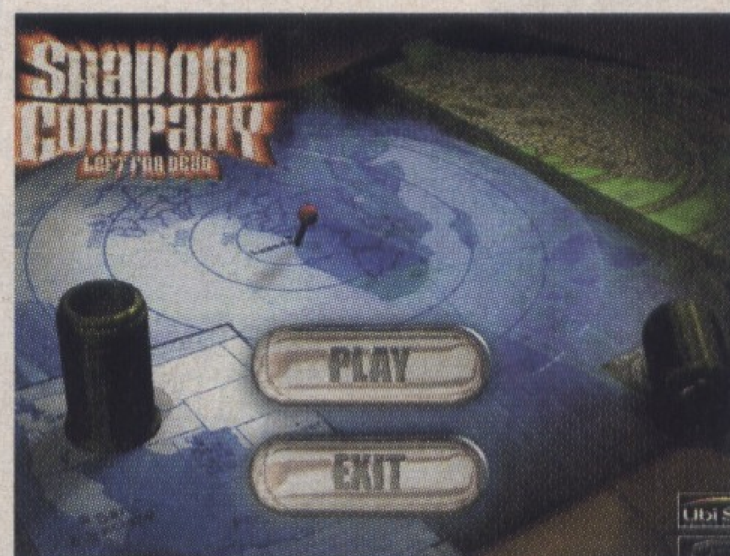
Proyectos cancelados

No todo es un camino de rosas en el sufrido mundo de los programadores de juegos. Los proyectos que se cancelan por diversas causas son multitud, y muchos con un presupuesto que haría brincar de alegría a cualquier compañía de desarrolladores españoles.

Los proyectos más sonoros que van a pasar al limbo del olvido van a ser la versión para PC de *Mission*





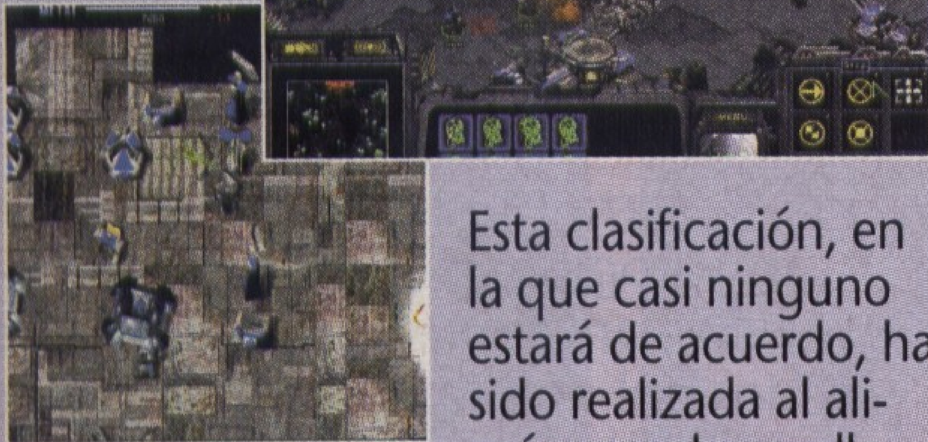




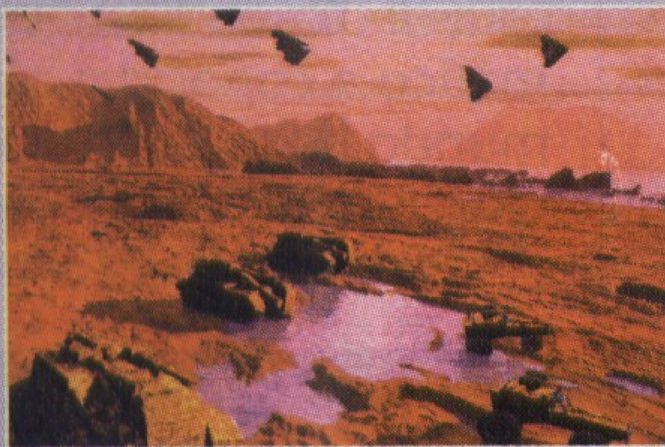
Impossible, debido a que el tiempo de desarrollo de este producto ha sido estimado como demasiado largo; *ShadowPact*, de Blue Byte, un juego de estrategia táctica en 3D, muy parecido a *Shadow Company* pero con argumento relativo al mundo del espionaje; *Amen: the Awakening* de Cavedog, creadores del juego *Total Annihilation*. En este último caso la cancelación se ha debido a problemas de retrasos acumulados. Otros juegos que no saldrán al mercado, a pesar de que ya se había empezado a trabajar en ellos son la nueva secuela de *Mortal Kombat* para la consola Nintendo 64, o Third World, un RPG táctico que no ha podido encontrar un editor.

Los 100 mejores de todos los tiempos

Y otra lista, esta vez los 100 mejores juegos de toda la historia y para todo tipo de plataformas. Parece que una vez agotado el segundo milenio hay que hacer recapitulación obligada de todos los títulos que se recordarán por su gran calidad, su jugabilidad o su originalidad.



Esta clasificación, en la que casi ninguno estará de acuerdo, ha sido realizada al alimón por desarrolladores de videojuegos y por redactores de la prestigiosa revista de software lúdico Edge. De lo que estamos seguros es que habrá pocas personas que hayan probado los 100 títulos, pero es bastante fácil que encuentres representado en algún lugar a tu juego o juegos favoritos.

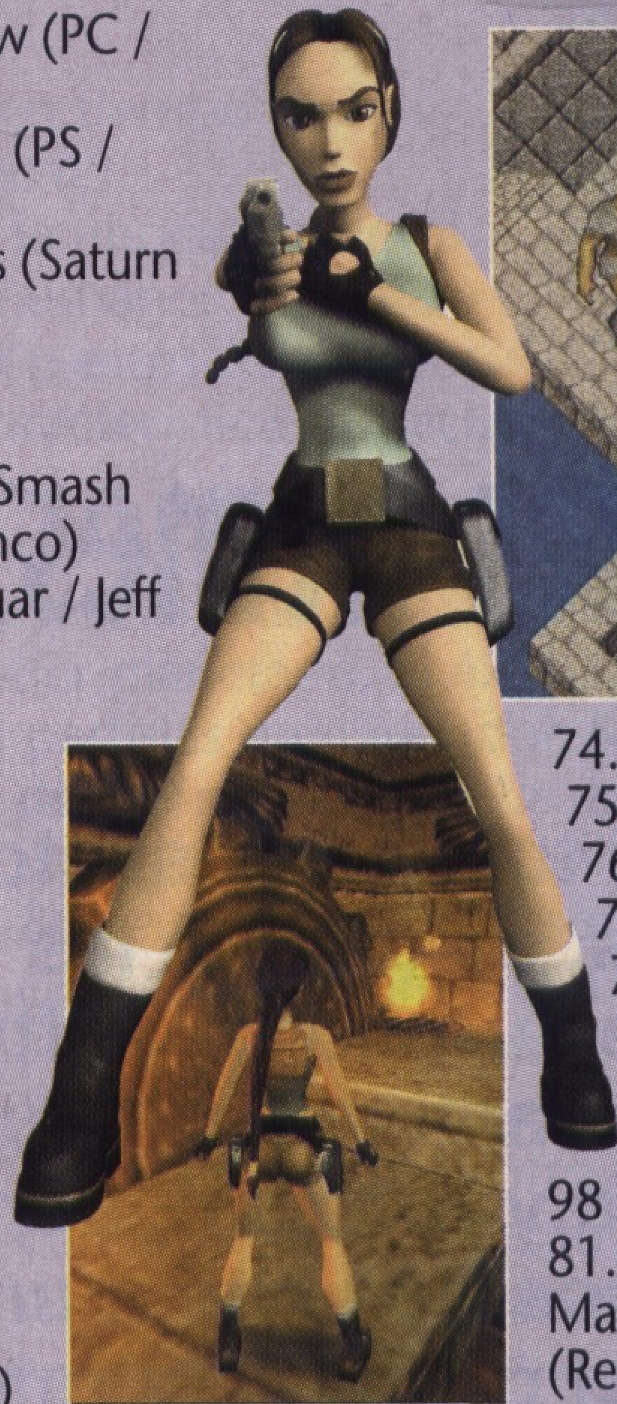


1.Legend of Zelda: Ocarina of Time (N64 / Nintendo)
2.Super Mario 64 (N64 / Nintendo)
3.Goldeneye (N64 / Rare)
4.Tetris (Gameboy / Bulletproof)
5.Gran Turismo (PlayStation / Polyphony Digital)
6.Quake II (PC / Id)
7.The Legend of Zelda: Link to the Past (SNES / Nintendo)
8.Street Fighter II Turbo (Recreativa / Capcom)
9.Half-Life (PC / Valve)
10.Super Mario World (SNES / Nintendo)
11.Pokemon (Gameboy / Game Freak)
12.Civilization 2 (PC / Microprose)
13.Final Fantasy VII (PS / Squaresoft)
14.Super Metroid (SNES / Nintendo)
15.Super Puyo Puyo (SNES / Compile)
16.Metal Gear Solid (PS / Konami)
17.Super Bomberman (SNES / HudsonSoft)
18.Super Mario Kart (SNES / Nintendo)
19.Tekken 3 (PS / Namco)
20.Tomb Raider (PS / Core)
21.Total Annihilation (PC / Cavedog)
22.GP2 (PC / Geoff Crammond)





23.X-Com Enemy Below (PC / Microprose)
24.Final Fantasy Tactics (PS / Squaresoft)
25.Nights: Into Dreams (Saturn / Sega Sonic Team)
26.Sim City 2000 (PC / Maxis)
27.Anna Kournikova´s Smash Court Tennis (PS / Namco)
28.Tempest 2000 (Jaguar / Jeff Minter)
29.Starcraft (PC / Blizzard)
30.Soul Calibur (Dreamcast / Namco)
31.Pro Evolution (PS / Konami)
32.Resident Evil (PS / Capcom)
33.Castlevania: Symphony of the Night (PS / Konami)
34.International Track and Field (PS / Konami)
35.PilotWings 64 (N64 / Nintendo Paradigm)
36.Colin McRae Rally (PS / Codemasters)
37.Bust-a-Move 2 (PS / Taito)
38.Doom II (PC / Id)
39.X-Wing vs TIE Fighter (PC / LucasArts)
40.Hidden & Dangerous (PC / Illusion)
41.1080 Snowboarding (N64 / Nintendo)
42.Secret of Mana (SNES / Squaresoft)
43.Sega Rally (Recreativa / Sega AM3)
44.Star Fox 64 (N64 / Nintendo)
45.Wave Race 64 (N64 / Nintendo)
46.Ridge Racer Type 4 (PS / Namco)
47.Ultima Online (PC / Origin)







48.Secret of Monkey Island (PC / LucasArts)
49.Robotron 2084 (Recreativa / Eugene Jarvis)
50.R-Type (Recreativa / Irem)
51.Sonic Adventure (DC / Sega Sonic Team)
52.Bubble Bobble (Recreativa / Taito)
53.OutRun (Sega / AM2)
54.Stunt Car Racer (Amiga / Geoff Crammond)
55.Speedball 2 (Amiga / Bitmap Bros)
56.Micro Machines 2 (Genesis / Codemasters)
57.Death tanks (Saturn / Zombie)
58.Daytona USA (Recreativa / Sega AM2)
59.Sensible Soccer (Amiga / Sensible)
60.Sonic the Hedgehog (Genesis / Sega)
61.Virtual Fighter 2 (Recreativa / Sega AM2)
62.The Need for Speed (3DO / EA)
63.Elite (BBC Model B / Acornsoft)
64.Defender (Recreativa / Eugene Jarvis)
65.Head Over Heels (Spectrum /Ritman–Drummond)
66.Super Sprint (Recreativa / Atari)
67.Mercenary (C64 / Paul Woakes)
68.The Sentinel (C64 / Geoff Crammond)
69.Exile (Amiga / Smith-Irving)





70.Grand Theft Auto (PC / DMA Design)
71.Shinobi (Recreativa / Sega)
72.Contra III (SNES / Konami)
73.Rampart (Recreativa / Atari)
74.Pac-Man (Recreativa / Namco)
75.Galaga (Recreativa / Namco)
76.Bomb Jack (Recreativa / Tehkan)
77.Return Fire (3DO / Studio 3DO)
78.Chu-Chu Rocket (DC / Sega Sonic Team)
79.Gauntlet II (Recreativa / Atari)
80.NHL 98 (PS / EA)
81.Marble Madness (Recreativa / Atari)







82.Populous 2 (Amiga / EA)
83.Super Pang (Recreativa / Mitchell)
84.Thrust (C64 / Jeremy Smith)
85.Legend of the Mystical Ninja (SNES / Konami)
86.Xenogears (PS / Square / EA)
87.Nemesis (Recreativa / Konami)
88.Lemmings (Amiga / DMA)
89.Asteroids (Recreativa / Atari)
90.Paradroid (C64 / Andrew Braybrook)
91.Boulderdash 2 (C64 / First Star)
92.Wipeout (PS / Psygnosis)
93.Strider (Recreativa / Capcom)
94.NBA Jam (SNES / Iguana)
95.Syndicate (Amiga / Bullfrog)
96.Phantasy Star III (Genesis / Sega)
97.Choplifter (C64 / Dan Jolin)
98.Time Crisis (Recreativa / Namco)
99.Solomon´s key (Recreativa / Tecmo)
100.Super Punch Out (SNES / Nintendo)







Havas

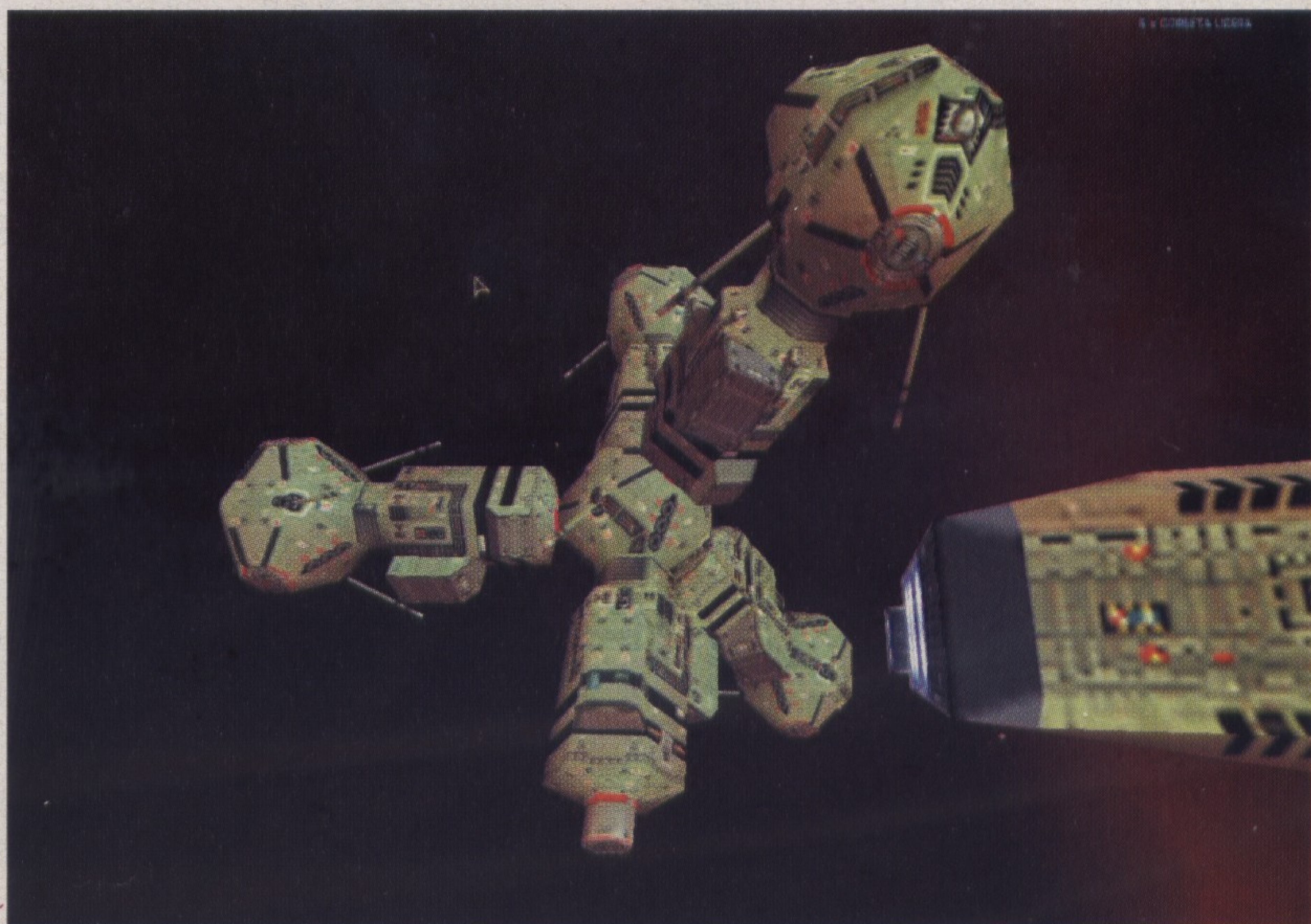
Gigante con pies firmes



Presente en casi todos los sectores del ocio mediático, Havas, es una de las empresas de distribución y desarrollo de software más importantes de la actualidad. Sus filiales más conocidas, Sierra y Blizzard, son auténticas factorías de juegos de éxito mundial. Vamos a dar un repaso del presente y futuro de esta compañía.

En el momento actual se puede decir sin temor a faltar a la verdad que Havas es una de las empresas más potentes en cuanto a desarrollo y distribución de todo tipo de software. Las actividades de esta compañía se dividen en cuatro grandes bloques:

Hay una división que se ocupa de desarrollar programas dedicados en su totalidad al sector educativo. Coktel y Knowledge Adventure son las encargadas del lanzamiento de materiales de complemento escolar, desde juegos educativos a programas que intentan despertar y potenciar las habilidades de los más pequeños. También producen diccionarios o métodos de aprendizaje de idiomas.



Sierra Home, se ocupa de ofrecer programas de edición y creación para profesionales y aficionados del diseño por ordenador. *Print Artist, Web Artist* o *Photo Artist* son sus trabajos más conocidos.

La tercera división de Havas está dedicada a dar soporte informático a conocidas obras del mundo editorial. Obras tan conocidas como la enciclopedia Larousse, el Pequeño Larousse o Kleio conocen ya una edición electrónica gracias a Havas.

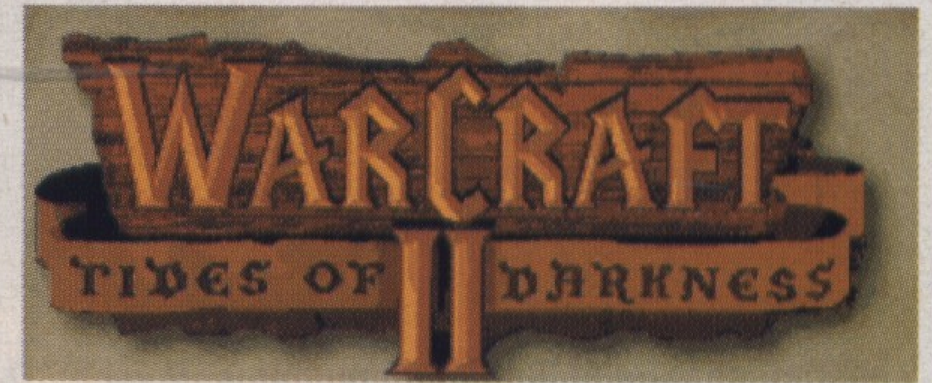




Pero lo que más nos interesa a nosotros es el segmento de software lúdico. Las conocidas marcas Sierra y Blizzard Entertainment son propiedad exclusiva de Havas. Las ventas de juegos tan conocidos como *Half-Life, King´s Quest, Caesar, Gabriel Knight, Faraón, Warcraft, Starcraft* o *Diablo*, son



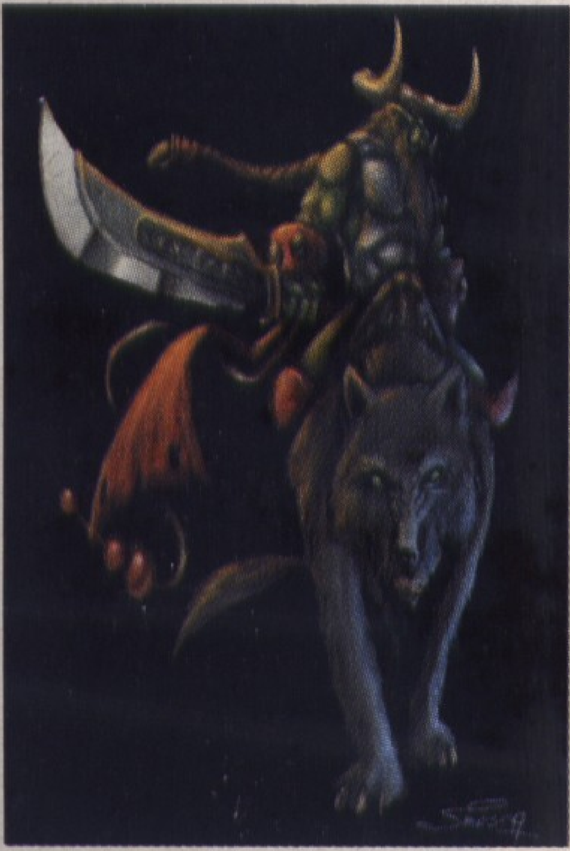
Warcraft y Starcraft



Cuando se hace un repaso a los clásicos de la estrategia en tiempo real es imprescindible mencionar a estos dos juegos: *Warcraft* y *Starcraft*.

En *Warcraft 1* y *2* los escenarios estaban ambientados en la época medieval. Eran dos estupendos títulos en los que dos bandos: orcos y humanos, luchaban denodadamente por el control del territorio. La perspectiva en estos dos juegos era isométrica y las unidades a comandar tenían cada una características especiales que había que combinar adecuadamente.





Starcraft, un verdadero bombazo en cuanto a ventas se refiere, tenía un estilo más futurista. Tres razas, con características muy diferentes luchaban por el poder planetario. El éxito del juego se baso en gran parte en las partidas multijugador. Más de 4,5 millones de usuarios se registraron en el servidor *Battle.net*, y aún hoy son miles las personas que se conectan diariamente a la Red para echar una partidita a este juego.



Warcraft III, todavía en pleno período de desarrollo, seguirá ubicado dentro del género estratégico, pero contará con un aspecto totalmente tridimensional, cosa cada vez más usual en los juegos de este tipo.

estratosféricas. Sólo como ejemplo puntual decir que de la saga *King´s Quest* se han vendido más de siete millones de copias en el mundo.

Con estos datos no es de extrañar que Havas Interactive, la filial de Havas que se ocupa del desarrollo y distribución de juegos de ordenador, sea el mascarón de proa de la compañía. Sólo esta división de Havas ocupa a más de tres mil personas en todo el mundo, de las cuales 430 están empleadas en Europa. Havas Interactive Europa tiene las oficinas principales en Francia, y las secundarias en Gran Bretaña, Alemania, España e Irlanda.

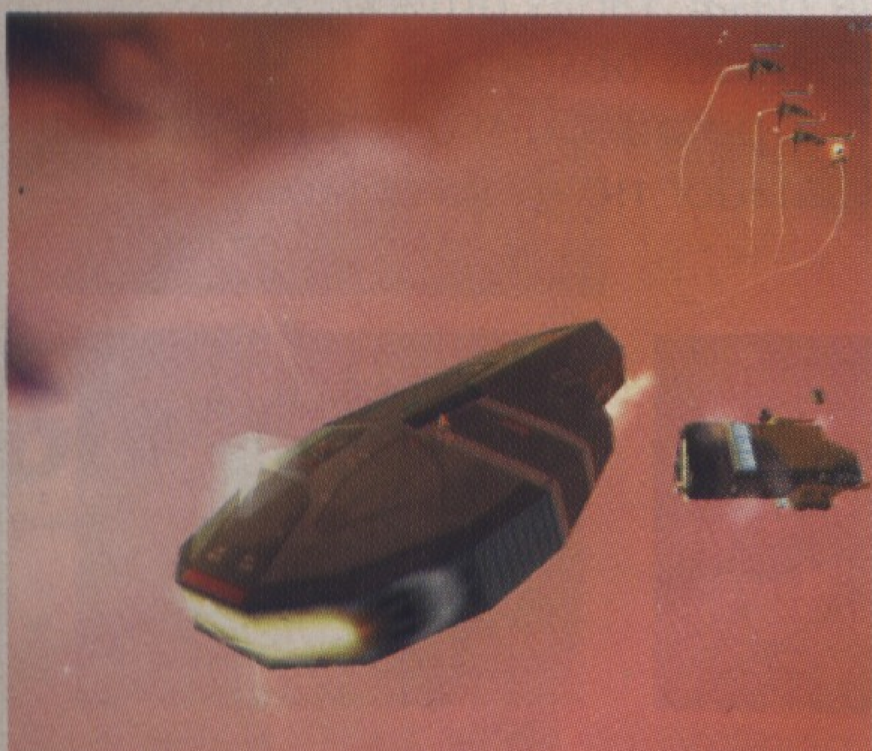


Los objetivos de futuro que se ha marcado Havas pasan casi todos por potenciar su presencia en Internet. Desde potenciar el software *Adi*, capaz de retransmitir clases virtuales a través de la Red, a expansionar la cartera de clientes que usan Internet para jugar partidas multijugador. El despliegue de servidores *Battle.net* o *Won.net*, permitirá a los usuarios de juegos de Havas contar con un amplio servicio de juegos en línea.

Las compañías filiales de Havas Interactive son las conocidísimas Sierra y Blizzard

Y por supuesto otro objetivo prioritario de esta empresa es seguir desarrollando títulos de van-





El señor de las Tinieblas



Diablo supuso una auténtica revolución en los juegos de rol. Acostumbrados como estabamos a títulos de este género donde la acción era más bien escasa, o por lo menos no la parte esencial del juego, *Diablo* imprimió un nuevo estilo a los juegos de rol, luego imitado hasta la saciedad en otros juegos de la competencia.



En *Diablo* aparte de recoger objetos, aprender hechizos, o explorar mazmorras, teníamos que acabar con todos los seres de pesadilla que nos encontrábamos a base de "clics" de ratón. Tremendamente adictivo.

Ahora se espera la inminente aparición de la segunda parte de este juego, *Diablo II*, que seguirá la misma línea que el primero de la saga y que, seguro, se convierte en superventas desde el mismo momento en que aparezca.







guardia que le permitan seguir liderando el mercado de software de entrenimiento. Los próximos lanzamientos de la compañía son esperados con expectación por los jugadores de todo el mundo. *Diablo 2, Warcraft 3, Team Fortress 2, Arcanum* o *Ground Control* están en plena fase de desarrollo o a punto de salir al mercado. Todos estos datos hacen augurar que habrá que seguir hablando de esta compañía en los próximos meses.

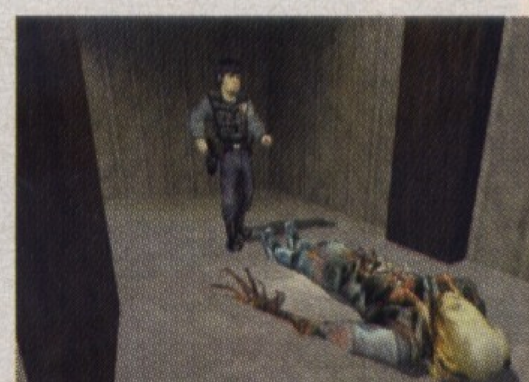
La compañía espera expansionar todos sus negocios que tengan algo que ver con Internet

Alfredo del Barrio

Disparando sin parar

Uno de los juegos más premiados, y también de los más vendidos, del año 99 fue *Half-Life.* Elegido juego del año por varias publicaciones del medio, consiguió colocar en los hogares de los aficionados a los juegos de ordenador la respetable cifra de más de un millón de ejemplares.

Este juego era un shoot´em up, es decir uno de esos juegos en perspectiva subjetiva en los que disparar rápido y con buena puntería es vital para conseguir avanzar a través de sus pantallas. Además, contaba con elementos que obligaban al jugador a discurrir un poco mientras cambiaba el vacío cargador de su arma favorita. La acción estaba situada en un laboratorio de experimentación en el que uno de los proyectos falla y deja abierta una puerta tridimensional por la que aparecen unas criaturas nada amigables. Sobrevivir se convierte en nuestro único objetivo.



Ahora podemos encontrar en las tiendas la expansión oficial de este juego, *Half-Life Opposing Force*, que cuenta con una campaña completa para un solo jugador, nuevos mapas multijugador, nuevas armas y nuevos personajes. Más que recomendable para el afortunado poseedor del juego original.









La Aventura con mayúsculas

Como podéis ver Havas Interactive tiene títulos clásicos en casi todos los géneros. En el segmento de las aventuras gráficas un juego mítico, por la gran solera que atesora, es la serie *King´s Quest*.

Esta saga lleva la friolera de ocho capítulos y revisarla entera es como ver el ejemplo perfecto de cómo han evolucionado gráficamente los juegos de ordenador, desde los diseñados para jugarse en los primeros 286 hasta los desarrollados para ser utilizados en los potentes Pentium actuales.

Aparte de las increíbles mejoras en el apartado gráfico, la mecánica de *King´s Quest* sigue siendo la misma: aventurarse por un mundo de fantasía donde para vencer al malo de turno tendremos que pasar varias pruebas en las que nuestra inteligencia y poder de concentración son esenciales para superar los numerosos obstáculos.

No se sabe nada todavía de cuándo aparecerá la novena entrega de esta saga épica, pero seguro que ya están trabajando en ella. Hacerse un hueco en la memoria colectiva de los jugadores de ordenador no es tarea fácil y no se puede desaprovechar un título famoso así como así. Seguiremos esperando.











más vendi-
arias publi-
los aficio-
ás de un
juegos en
ena pun-
llas.
or a discu-
rma favo-

entu-
evo-
s
ven-
as
án
no

Inteligencia Artificial

El gran desafío

En este número vamos a tratar uno de los temas más oscuros y menos conocidos de toda la informática, la inteligencia Artificial o IA. Si algún día queremos realizar algún programa que la use, primero deberemos entender qué es, su historia y cómo ha evolucionado a lo largo del tiempo. En este artículo os vamos a hablar de las tres cosas.

Comenzamos con una breve historia: la Inteligencia Artificial "nació" en 1943 cuando Warren McCulloch y Walter Pitts propusieron un modelo de neurona del cerebro humano y animal. Estas neuronas nerviosas informáticas proporcionaron una representación simbólica de la actividad cerebral. Algo después, Norbert Wiener tomó estas y otras ideas y las elaboró dentro de un mismo campo que se llamó "Cibernética". De aquí nacería, sobre los años 50, la Inteligencia Artificial.

Los primeros investigadores de esta innovadora ciencia, tomaron como base la neurona formalizada de McCulloch y, con ella, postularon la teoría de que *el cerebro es un solucionador inteligente de problemas, de modo que imitemos al cerebro.*

El cerebro es un solucionador inteligente de problemas, de modo que imitemos al cerebro



Entra al futuro, recuerda el pasado, dice el lema de esta asociación americana sobre IA.

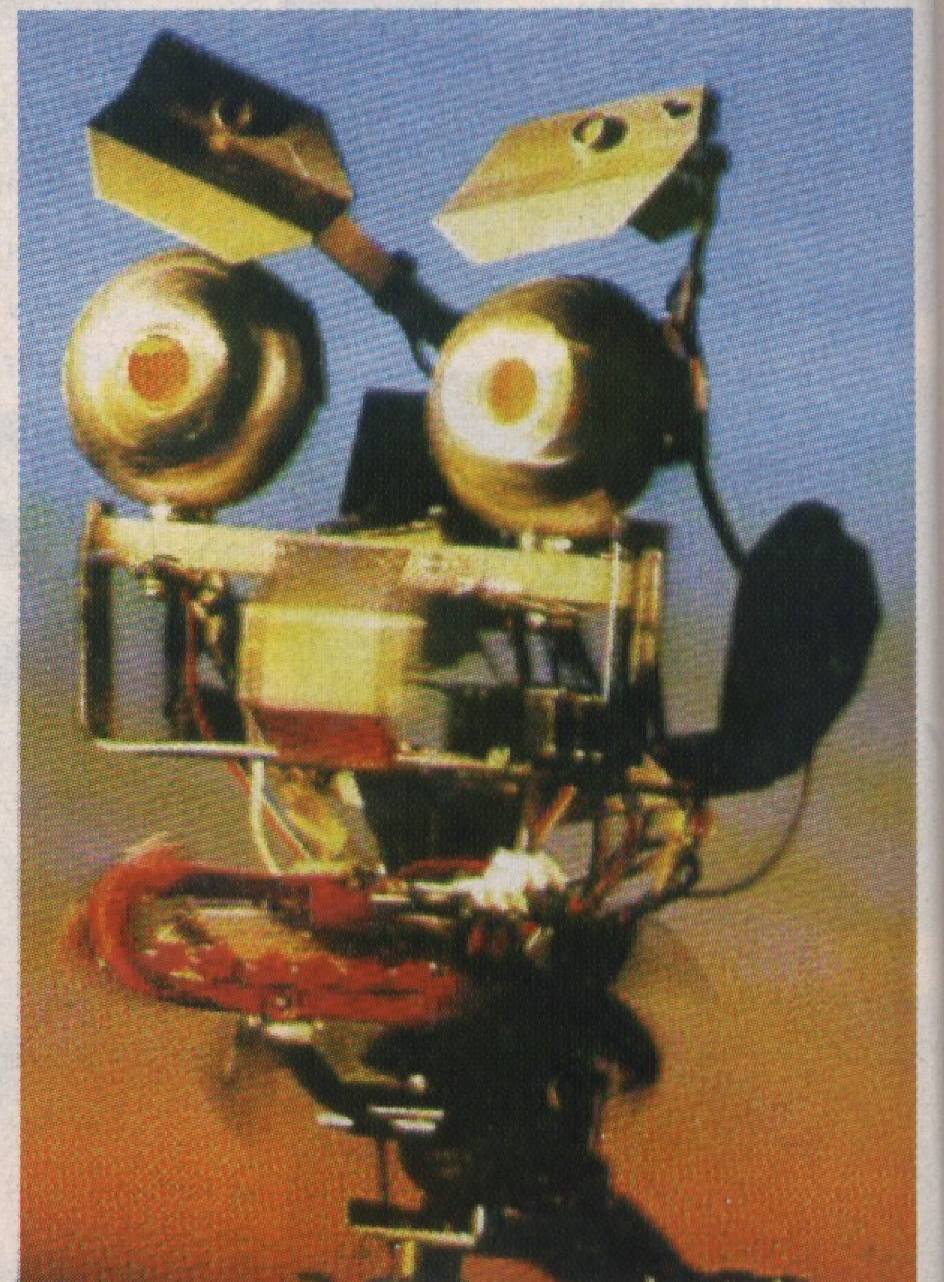
Esta era la base fundamental; sin embargo, si consideramos la enorme complejidad del cerebro, esto es algo prácticamente imposible hoy día de modo que ni mencionar que ni el hardware ni el software de la época estaban a la altura de las circunstancias para realizar semejantes proyectos.

Por lo tanto, podemos decir a grandes rasgos que la Inteligencia Artificial es una ciencia que intenta la creación de programas para máquinas que imiten el comportamiento y la comprensión humana, que sea capaz de aprender, reconocer y pensar. Es en los años 50 cuando se logra realizar un sistema que tuvo cierto éxito. Se llamó el *Perceptrón* de Rossenblatt. Éste era un sistema visual de reconocimiento de patrones en el cual se aunaron esfuerzos para que se pudieran resolver una gama amplia de problemas, pero estas energías se diluyeron enseguida.

En los años 70, un equipo de investigadores dirigido por Edward Feigenbaum comenzó a elaborar un proyecto para resolver problemas de la vida cotidiana o que se centrara, al menos, en problemas concretos. Así es como nació el sistema experto.

El primer sistema experto fue el denominado Dendral, un intérprete de espectrograma de masa. Pero el más influyente resultaría ser el *Mycin* de 1974. Éste era capaz de diagnosticar trastornos en la sangre y recetar la correspondiente medicación, todo un logro de aquella época que incluso llegó a utilizarse en hospitales.

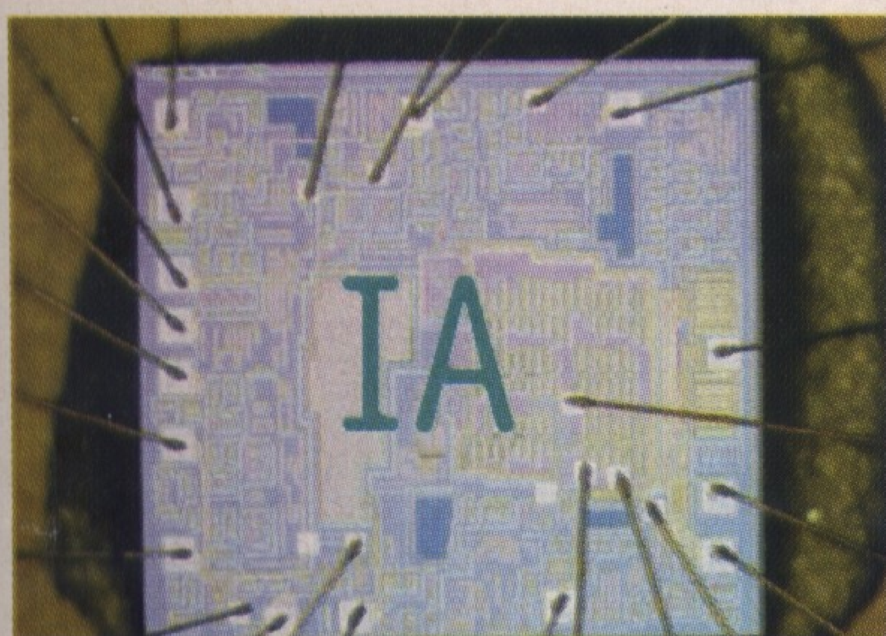
Ya en los años 80, se desarrollaron lenguajes especiales para utilizar con la


Este robot es capaz de resolver problemas externos poco complejos mediante el uso de IA.

Inteligencia Artificial, tales como el *LISP* o el *PROLOG*. Es en esta época cuando se desarrollan sistemas expertos más refinados, como por el ejemplo el *EURISKO*. Este programa perfecciona su propio cuerpo de reglas heurísticas automáticamente, por inducción; es decir: es capaz de "aprender".

Tomando como definición de Inteligencia Artificial, la que dio en su día Marvin Minsky, *la Inteligencia Artificial es el arte de hacer máquinas capaces de hacer cosas que requerirían inteligencia en caso de que fuesen hechas por seres humanos,* podríamos entender (o al menos nos ayudaría a hacerlo) por qué en inteligencia artificial hay tantas ramas: al igual que en medicina, la I.A. debe abarcar todo lo referente al hombre, para podérselo ofrecer a la máquina.

Ciertamente, cuando aparecieron por primera vez los ordenadores, la mayoría de sus diseñadores los proyectaron para que no hicieran nada aparte de enormes cálculos sin discernimiento. De ahí es de donde viene la palabra *computers*, del verbo *to comput*, es decir, calcular. Pero incluso entonces, unos pocos pioneros, especialmente Alan Turing (también pionero en Vida Artificial) vislumbraron ya lo que ahora se conoce como inteligencia artificial. Estos investigadores vieron que los ordenadores podrían ir más allá de la aritmética y quizás imitar los procesos que tienen lugar en el interior del cerebro humano.



La IA o AI (Artificial Inteligence) en ingles, es una ciencia poco conocida todavía, en la que hay muchos excépticos.

Alan Turing, uno de los más grandes matemáticos de todos los tiempos, fue el que lideró a un grupo de expertos para descifrar el código alemán <<*Enigma*>>, que ayudó de forma decisiva a la victoria del bando aliado en la Segunda Guerra Mundial. Una vez acabada la guerra, Turing escribió el primer programa que ya se podría decir que usaba Inteligencia Artificial, esta maquina era un simple juego de Ajedrez.

Resolución de problemas

Difícilmente podemos esperar ser capaces de hacer que las máquinas hagan maravillas antes de descubrir cómo hacer que hagan cosas normales y sensatas. Los primeros programas de ordenador eran poco más que simples listas y bucles de comandos con condiciones como *haz esto si ocurre aquello* que lo único que hacían era ejecutar una serie de sentencias. La mayoría de la gente todavía escribe programas en aquellos lenguajes (como BASIC) que obligan a imaginar todo lo que el programa hará desde un momento al siguiente. Llamemos a este tipo de programación "haz esto".

Poco después, los investigadores de IA encontraron nuevas formas de crear programas. En su sistema *General Problem Solver* (solucionador general de problemas), construido a finales de los años 50, Allen Newell, J.C.Shaw y Herber A. Simon mostraron formas de describir procesos en términos de afirmaciones como *si la diferencia entre lo que tienes y lo que quieres es de tipo D, entonces intenta cambiar dicha diferencia usando el método M.*

Ésta y otras ideas condujeron a lo que llamamos métodos de programación "significados-objetivos" y "hacer si es necesario". Tales programas aplican automáticamente instrucciones en el momento en que se necesitan, con lo que los programadores no tienen que anticipar en qué momento ocurrirá eso. Esto empezó una era de programas que podían resolver problemas de formas que sus programadores no podían anticipar, porque a los programas se les podría haber dicho qué tipo de cosas intentar sin saber por anticipado cuál funcionaría.

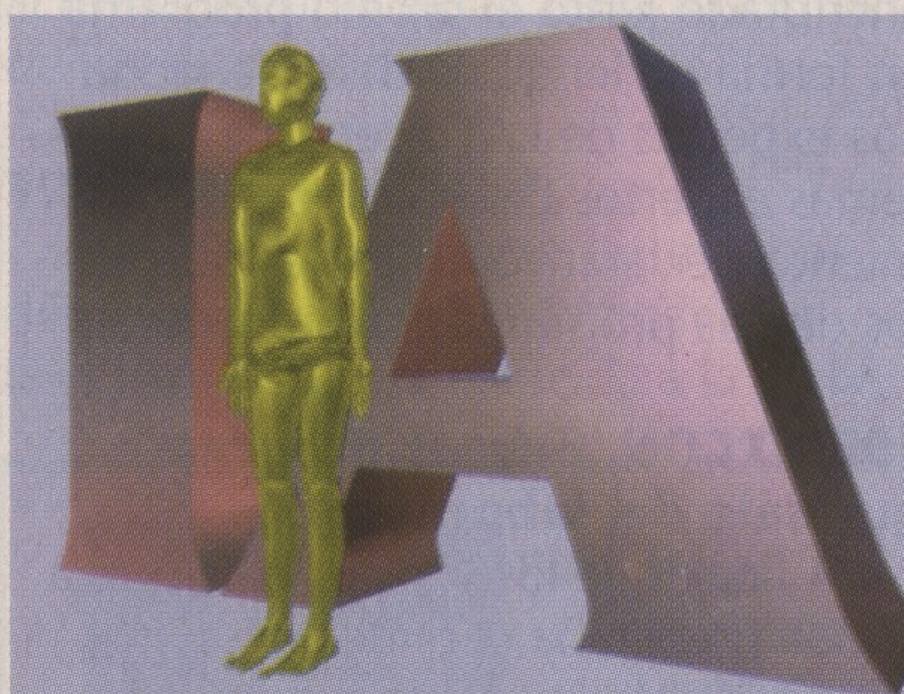
Todo el mundo sabe que si se intentan suficientes cosas diferentes aleatoriamente, finalmente se puede hacer cualquier cosa. Los nuevos sistemas no hacían cosas aleatoriamente sino que recibían "consejos" sobre lo que era más probable que funcionara con cada tipo de problema. Así, en vez de vagar al azar, tales programas podían abrirse camino en cierta forma. El único problema era una tendencia a quedarse atascados en picos más pequeños.

Desde entonces, muchas de las investigaciones sobre IA se han dirigido a la búsqueda de métodos más "globales" para superar las distintas formas de quedarse atascados, haciendo para ello programas que tuviesen perspectivas más amplias y planificasen por adelantado. A pesar de todo, nadie ha descubierto una forma "completamente universal" de encontrar siempre el mejor método y, podemos añadir, nadie espera hacerlo.

En su lugar, muchos investigadores de IA aspiran hoy día a obtener programas que establezcan equivalencias entre patrones en la memoria para decidir qué hacer después.

Hay que pensar en esto como programación "haz algo sensato". Unos pocos investigadores experimentan con programas que pueden aprender y razonar por analogía. Estos programas algún día reconocerán cuáles de sus antiguas experiencias en memoria son más análogas a las nuevas situaciones, de forma que puedan "recordar" qué métodos funcionaron mejor con problemas parecidos en el pasado y utilizarlos.

Los programas de ordenador más comunes hacen sólo las cosas para las que están programados. Algunos programas de IA son más flexibles; cuando algo sale mal pueden retroceder hasta algunas de las decisiones previas e intentar algo más. Pero incluso eso es una base demasiado tosca para la inteligencia. Para hacerlos verdaderamente inteligentes tendremos que hacerlos más reflexivos. Cuando las cosas salen mal, una persona intenta comprender qué es lo que está equivocado en lugar de simplemente intentar otra cosa. Buscamos explicaciones causales o excusas y, cuando las encontramos, las añadimos a nuestras redes de creencias y comprensión. Hacemos un aprendizaje inteligente. Algún día los programas podrían hacer también esas cosas, pero primero necesitaríamos realizar muchas más investigaciones para averiguar cómo.



La finalidad de la IA, sin duda alguna, es llegar a simular el pensamiento y razonamiento humano.



Las principales ramas en que se divide la IA son la simbólica y la subsimbólica.

Pero hay otras definiciones más aceptadas: la *IA trata de construir máquinas con comportamiento aparentemente inteligente. El hombre es un ser inteligente*. Surgen los dos grandes bloques enfrentados en la materia: el enfoque simbólico o *Top-Down*, también llamado IA clásica, y el enfoque subsimbólico o *Bottom-Up*, llamado a veces conexionista.

> Alan Turing lideró a un grupo de expertos para descifrar el código alemán <<Enigma>>

Los simbólicos simulan directamente las características inteligentes que se pretenden conseguir. Como modelo de mecanismo inteligente a imitar, lo mejor que tenemos y más a mano es el ser humano. Desde este punto de vista, poco interesa simular los razonamientos de los animales, y mucho menos simular procesos celulares. El boom de los Sistemas Expertos, ahora de capa caída, vino de la mano de este planteamiento.

Principales corrientes

Como ya hemos comentado son la corriente simbólica y la subsimbólica. Para los constructores de sistemas expertos, es fundamental la representación del conocimiento humano y debemos a ellos los grandes avances en este campo. Realizando una gran simplificación, se debe incluir en un sistema experto dos tipos de conocimiento: "conocimiento acerca del problema particular" y "conocimiento acerca de cómo obtener más conocimiento a partir del que ya tenemos".

Para el primero existen técnicas como los *Frames* (marcos) que fueron los padres de lo que hoy conocemos como "Programación Orientada a Objetos". El segundo es llamado

también mecanismo de inferencia y requiere además de un método de búsqueda que permita tomar decisiones, como por ejemplo, seleccionar la regla a aplicar del conjunto total de posibles reglas, siempre tomando la mejor regla posible en cada caso, haciendo la mejor selección. Esto puede parecer lo más sencillo, pero suele ser lo más difícil. Se trata de elegir y elegir bien, pero sin demorarse varios millones de años en hacerlo.

Como ejemplo representativo de la rama simbólica llevada al extremo, tenemos el proyecto Cyc de Douglas B. Lenat, con un sistema que posee en su memoria millones de hechos interconectados. Según Lenat, la inteligencia depende del número de reglas que posee el sistema, y *casi toda la potencia de las arquitecturas inteligentes integradas provendrá del contenido, no de la arquitectura.* Para él, los investigadores que esperan poder resolver con una única y elegante teoría todos los problemas de inferencia y representación de conocimientos, padecen celos de la física: ansían una teoría que sea pequeña, elegante, potente y correcta.

Los sistemas expertos forman parte de un firme y verdadero avance en inteligencia artificial al poder incorporar miles de reglas

Los esfuerzos de la otra rama de la IA, los subsimbólicos, se orientan a simular los elementos de más bajo nivel que componen o intervienen en los procesos inteligentes, con la esperanza que de su combinación emerja de forma espontánea el comportamiento inteligente. Los ejemplos más significativos probablemente sean las Redes Neuronales Artificiales y los Algoritmos Genéticos. Aunque parezcan un fenómeno reciente, estos paradigmas no son más jóvenes que los Sistemas Expertos de la IA clásica; simplemente tuvieron menor publicidad y financiación. En cualquier caso, pasaron desapercibidos.

El Primer modelo de red neuronal fue propuesto en 1943 por McCulloch y Pitts. El Perceptrón de Rosenblat apareció en 1959, produciendo una gran y breve expectación que quedó pronto en el olvido, y J.



Todos los programas de deportes como PCFutbol, FIFA, etc. utilizan IA para controlar a los jugadores.



El mejor sistema de IA es el sistema experto, que intenta imitar el aprendizaje humano, es decir, al cerebro.

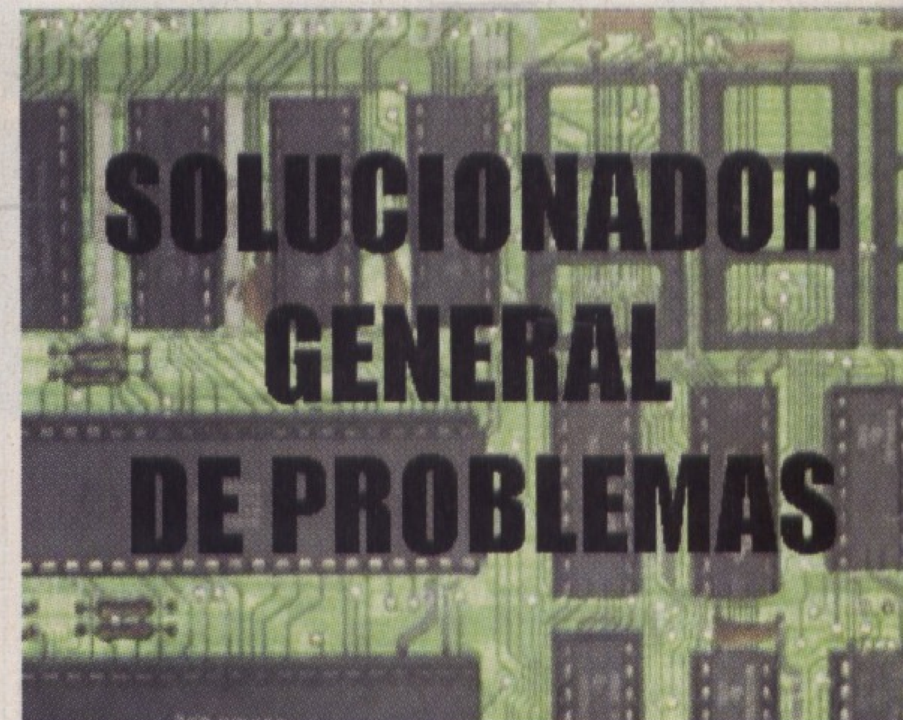
H. Holland introdujo la idea de los Algoritmos Genéticos en los años sesenta. Las grandes ventajas de estos sistemas son la autonomía, el aprendizaje y la adaptación, conceptos todos ellos relacionados.

¿Qué es un sistema experto?

Los sistemas expertos forman parte de un firme y verdadero avance en inteligencia artificial. Los sistemas expertos pueden incorporar miles de reglas. Para una persona sería una experiencia casi "traumática" el realizar una búsqueda de reglas posibles al completado de un problema y concordar éstas con las posibles consecuencias, mientras que se sigue en un papel los trazos de un árbol de búsqueda. Los sistemas expertos realizan amablemente esta tarea; mientras que la persona responde a las preguntas formuladas por el sistema experto, este busca recorriendo las ramas más interesantes del árbol, hasta dar con la mejor respuesta al problema, o en su falta, la más parecida a esta. Los sistemas expertos tienen la ventaja, frente a otros tipos de programas de Inteligencia Artificial, de proporcionar gran flexibilidad a la hora de incorporar nuevos conocimientos, es decir, de aprender. Para ello sólo tenemos que introducir la nueva regla que deseemos hacer constar y ya está, sin necesidad de cambiar el funcionamiento propio del programa. Los sistemas expertos son "autoexplicativos", al contrario que los programas convencionales, en los que el conocimiento como tal está encriptado junto al propio programa en forma de lenguaje de ordenador. Los expertos de I.A. dicen que los sistemas expertos tienen un conocimiento declarativo, mientras que en los demás programas es procedural.

1979 XCON, primer programa que sale del laboratorio

A finales de los sesenta la *Canergie Mellon* preparaba el principio de la insertación de la I.A. en el mundo real. El programa empezó llamándose R1, nombre que provenía de un malísimo chiste del creador del proyecto,

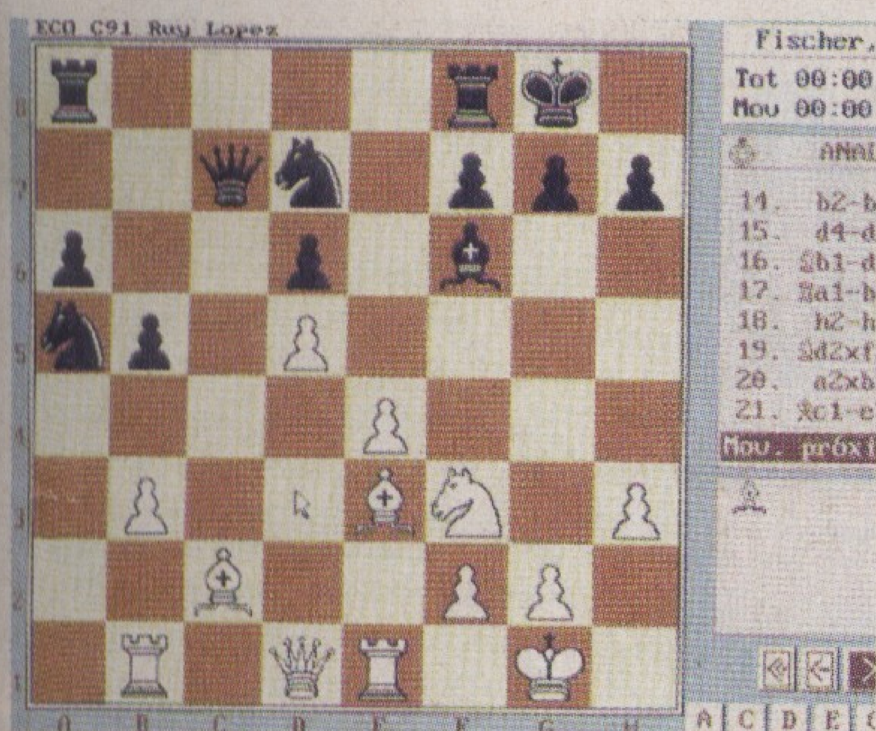


Este es el sistema de IA que se utiliza para que el ordenador sepa elegir cuál es la mejor solución a un problema.

pero la Digital Equipament Corporation, que era el usuario final de este programa, expresó su preferencia por otro nombre con más sentido, como el que al final quedó elegido: *XCON (eXpert CONfigurer* o Configurador experto).Cuando fue exigida la ayuda de McDermott, la DEC iba a lanzar al mercado una nueva serie de ordenadores, los llamados *VAX*. Dado que todos los ordenadores tenían configuraciones distintas entre sí, la *VAX* estaba previendo el enorme culo de botella (nombre que se le da en informática a los bloqueos por colapso de información para procesar que se dan en segmentos determinados) que se iba a formar, cuando (como ya había pasado anteriormente con otra serie de ordenadores) los ordenadores tuviesen fallos de configuración y hubiese que arreglar este problema uno por uno, con el consiguiente gasto de tiempo y dinero que eso suponía. Fue entonces cuando alarmados los directivos de la DEC pensaron en poner en marcha una solución urgente! contratando a John McDertmott. El cometido del *XCON* sería, en definición, muy simple: se trataba de configurar todos los ordenadores que saliesen de la DEC. El informe de viabilidad de McDermontt expuso resultados prometedores, y en diciembre de 1978 se empezó a trabajar en el proyecto.

En abril de 1979 el equipo de investigación que lo había diseñado (con McDermott a la cabeza), pensó que ya estaba preparado para salir y "conocer el mundo", fue entonces cuando se hizo una prueba real, esperando resolver positivamente un 95% de las configuraciones, este porcentaje tan alto anhelado por McDermontt y compañía se quedó en un 20% al ser contrastado con la realidad; *XCON* volvió al laboratorio, donde fue revisado y a finales de ese mismo año (1979) funcionó con resultados positivos en la DEC.

En 1980 *XCON* se instauró totalmente en DEC. Y en 1984, el *XCON* había crecido hasta multiplicarse por diez. En 1986 la compañía había invertido más de cincuenta



La parte más importante y fundamental de cualquier juego de ajedrez es la IA.

años/hombre en el programa, pero se estaba recuperando con creces de su inversión al ahorrarse cuarenta millones de dólares al año.

1980-85: la revolución de los sistemas expertos

XCON significó el pistoletazo de salida para los sistemas expertos, convenciendo a las más importantes empresas a invertir en programas de este tipo. Incluso para los más conservadores y los radicales "Anti-IA", algo se hacía evidente: mientras que los costes de desarrollo de los sistemas expertos se hacían cada vez más bajos, el salario de los expertos humanos se hacía más alto. La formación de un futuro experto costaba mucho dinero y años de esfuerzo, mientras que una vez creado un sistema experto podía ser copiado y distribuido tantas veces como necesario fuese. Además, el sistema experto nunca se cansa, no necesita dormir, no se distrae, no se va a la competencia, ni se pone enfermo, ni se jubila, ni pide aumento de sueldo...

Muchas grandes empresas invirtieron en el campo de la I.A. creando grupos especializados en ésta, para que desarrollaran aplicaciones para la empresa.

Empresas como DEC, Xerox, Schlumberger-Doll y Texas Instruments gastaron un total (más 150 empresas más) de mil millones de dólares en el desarrollo de la I.A.

A raíz de todos estos adelantos e inversiones puestos en marcha desde la calle, es decir, fuera de los laboratorios de las universidades, como era costumbre hasta la fecha... ocurrió que la sociedad empezase a oír cosas sobre qué es la tal "Inteligencia Artificial", que llegaba a ocupar programas de televisión, y artículos en revistas científicas de todo tipo. Edward Feigenbaum fundó la Teknowledge Inc, pero esta empresa no fue la única, sino que le acompañaron muchas más, como: Carnegie Group, Symbolics, Lisp Machines Inc, Thinking Machines Corporation, Cognitive Systems Inc...formando una inversión total de 300 millones de dólares. Los productos más importantes que creaban estas nuevas compañías eran las "máquinas *Lisp*". Se trataba de unos ordenadores que ejecutaban programas *LISP* con la misma rapidez de un ordenador central. Y el otro producto eran las "herramientas de desarrollo de sistemas expertos", también llamados *shells* (conchas).

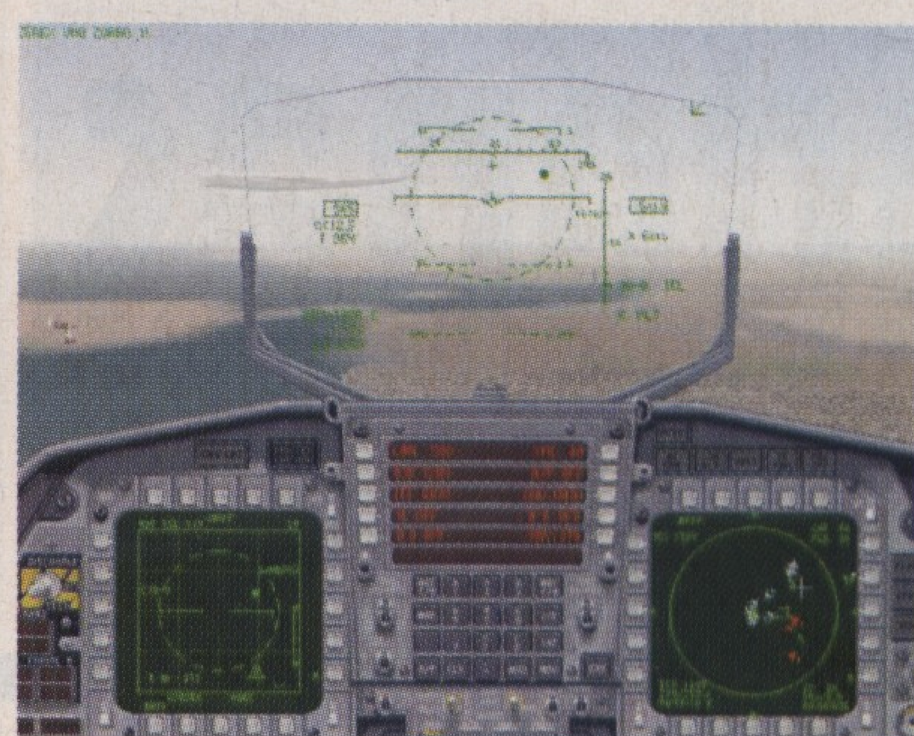
Este crecimiento, "desmesurado" a los ojos de los investigadores de I.A. más veteranos, provocó un colapso en la comunidad científica de I.A. haciendo que el primer congreso de la AAAI (Asociación Americana de Inteligencia Artificial) que se realizó en Stanford, en 1985, reuniese a más de mil investigadores. Y en la *International Joint Conference on Artificial Intelligence* (IJCAI) se llegó a las seis mil personas. Todo estaba cambiando para esos investigadores de I.A. que estaban acostumbrados a las "pequeñas" reuniones, casi familiares, en las que se podían hacer charlas, en las que la ropa era totalmente informal... Estas conferencias eran entonces (1985) el centro de reunión de empresarios, reporteros, científicos de otras ramas con curiosidad... Pero la demanda por parte de las empresas privadas de personal era mucho mayor a la oferta, esto hacia que los sueldos aumentasen enormemente y obligaban a los estudiantes a dejar la universidad cuanto antes. El MIT tenia en 1975 unos cuarenta investigadores y pasó a tener doscientos a finales de los 80.

1987: XCON empezó a crujir sobre su propio peso

Las constantes mejoras y cambios en los productos del DEC habían hecho que *XCON* se actualizara muy rápidamente, siendo objeto de la introducción de más y más reglas de funcionamiento, hasta llegar a tener más de diez mil reglas. A partir de ese momento, *XCON* no era mejor sino más grande, y DEC tuvo que gastarse dos millones de dólares al año para actualizar la base de conocimiento. Hasta esa fecha nadie se dio cuenta de que los sistemas expertos necesitaban un mantenimiento, y éste constituye hoy en día una de las actividades principales de la industria de los sistemas expertos.

1987: El fin del LISP

En 1987 aparecieron los microordenadores Apple y los compatibles IBM, con potencia parecida a la de las máquinas *LISP*, lo que hizo ver que no se necesitaban máquinas monotemáticas como éstas últimas, que además eran muy caras. Llegaron nuevas compañías de software como Gold Hill Computers y First Class Expert Systems, que transfirieron el software de I.A. a las máquinas convencionales utilizando el lenguaje C, lo que acabó totalmente con el *LISP*. Este último hizo que miles de personas perdieran el trabajo, dado que estos negocios no se practicaban desde universidades, sino desde empresas privadas.



Todos los simuladores de combate aéreos o terrestres llevan un sistema de IA.

En la actualidad:

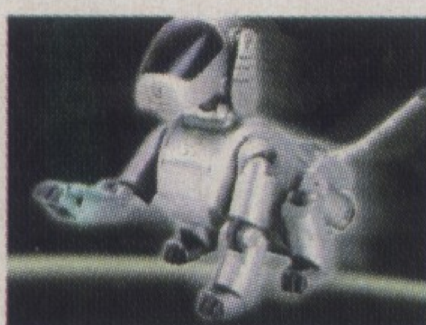
Hoy en día la IA es una parte fundamental de la informática, aunque mucha gente no la tome en serio, nos podemos encontrar ejemplos de IA en casi todos los juegos que existen hoy en el mercado, por ejemplo, sobre todo y fundamentalmente, en los de estrategia.

> Hoy en día la IA es una parte fundamental de la informática, aunque mucha gente no la tome en serio, nos podemos encontrar ejemplos de IA en casi todos los juegos que existen hoy en el mercado por ejemplo

Pero a través de la IA también se intenta analizar los procesos que tienen lugar en el interior del cerebro humano para poder comprender mejor el funcionamiento de éste. Cientos de robots diseñados por humanos usan este tipo de sistema para simular comportamientos y aptitudes que pudiesen imitar o parecer humanas. Incluso se usa en el cine.

En definitiva que la IA es y seguirá siendo una parte fundamental de la informática, aunque siempre oscura, misteriosa e ignorada por muchos, tratada como una ciencia sin sentido por otros, esta ciencia seguirá evolucionando como lo han hecho todas.....

Daniel García Alonso (PORTOX)
Dagal@Alehop.com



El Perro-Robot Autómata

Este robot fue diseñado por una compañía para imitar el comportamiento de un perro, es capaz de reaccionar a determinados estímulos mediante IA, es capaz de dar la pata, acurrucarse o comunicarse con sus amos.

Cadenas de texto

Tipos de datos "string"

El manejo de cadenas de texto era una de las carencias importantes de DIV1, pero con la llegada de DIV2 y sus funciones de manipulación de *strings*, se han solucionado en gran parte los problemas y además resulta muy sencillo.

Para poder manejar las cadenas de texto usamos el tipo de dato string. La declaración de un *string* es muy sencilla.

Si queremos declarar el *string* con la longitud por defecto (256 caracteres) y con todo su contenido puesto a "":

string <nombre>;

Para declarar el *string* con una determinada cadena de texto y con la longitud predeterminada (256 caracteres):

string <nombre> = "texto";

Para declarar el *string* con una determinada longitud:

string[longitud] <nombre>;

Si queremos indicar la longitud y el contenido de éste:

string[longitud] <nombre> = "texto" ;

Si el texto que ponemos es mayor que la longitud de la cadena el compilador dará un error avisando que el literal es demasiado largo.

Como los *strings* son *arrays* de caracteres podemos acceder individualmente a cada uno de ellos. Si quisiésemos mirar el primer carácter del *string* simplemente accederíamos a él mediante *texto [0]*. Recordemos que, en las tablas, el primer espacio es siempre el 0 y no el 1.

Funciones para el manejo de cadenas de texto

Hay varias. La primera que vamos a ver es la llamada: *char (<literal>)*.

H	o	l	a	\0

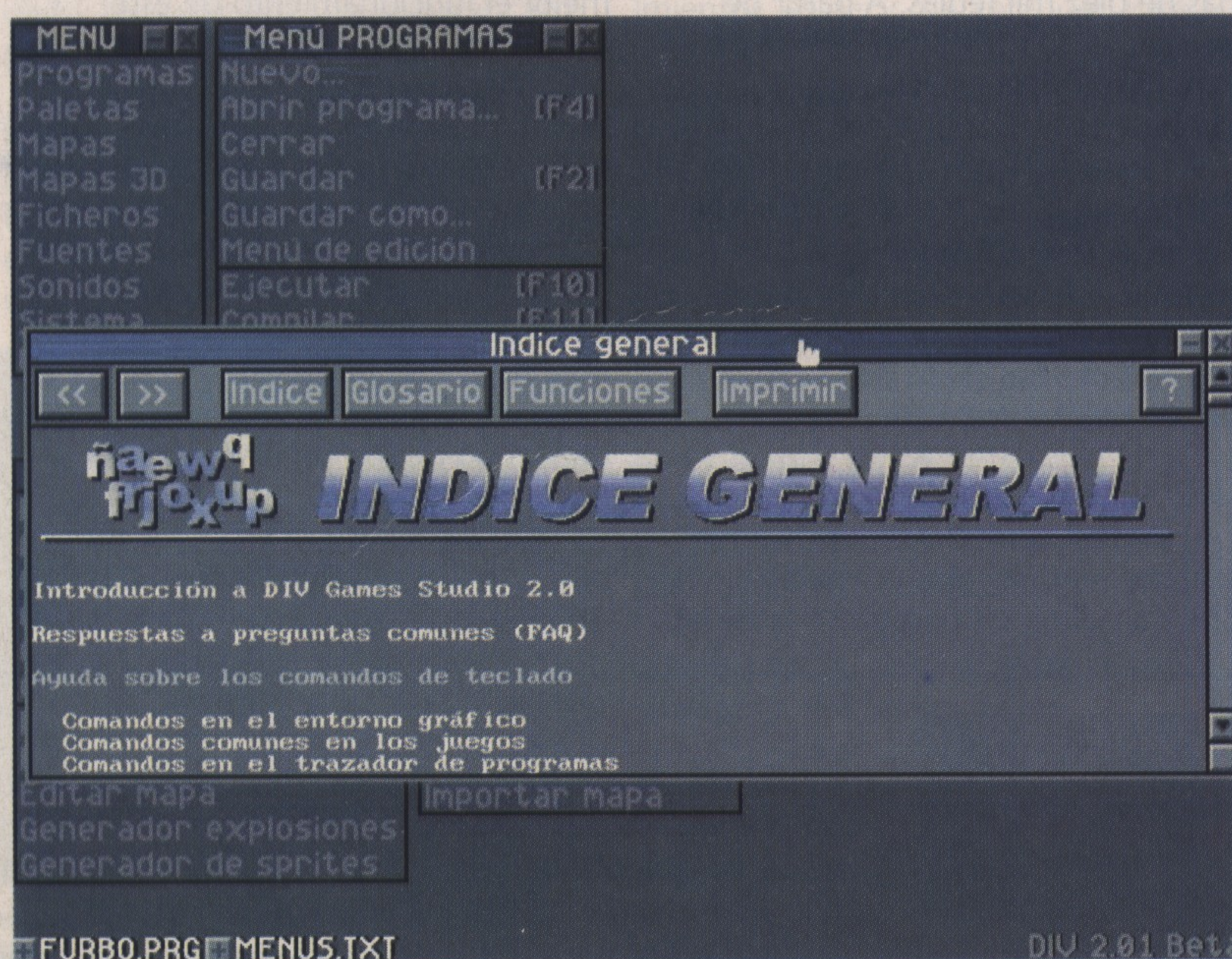
Las cadenas de texto finalizan con el carácter nulo (\0)

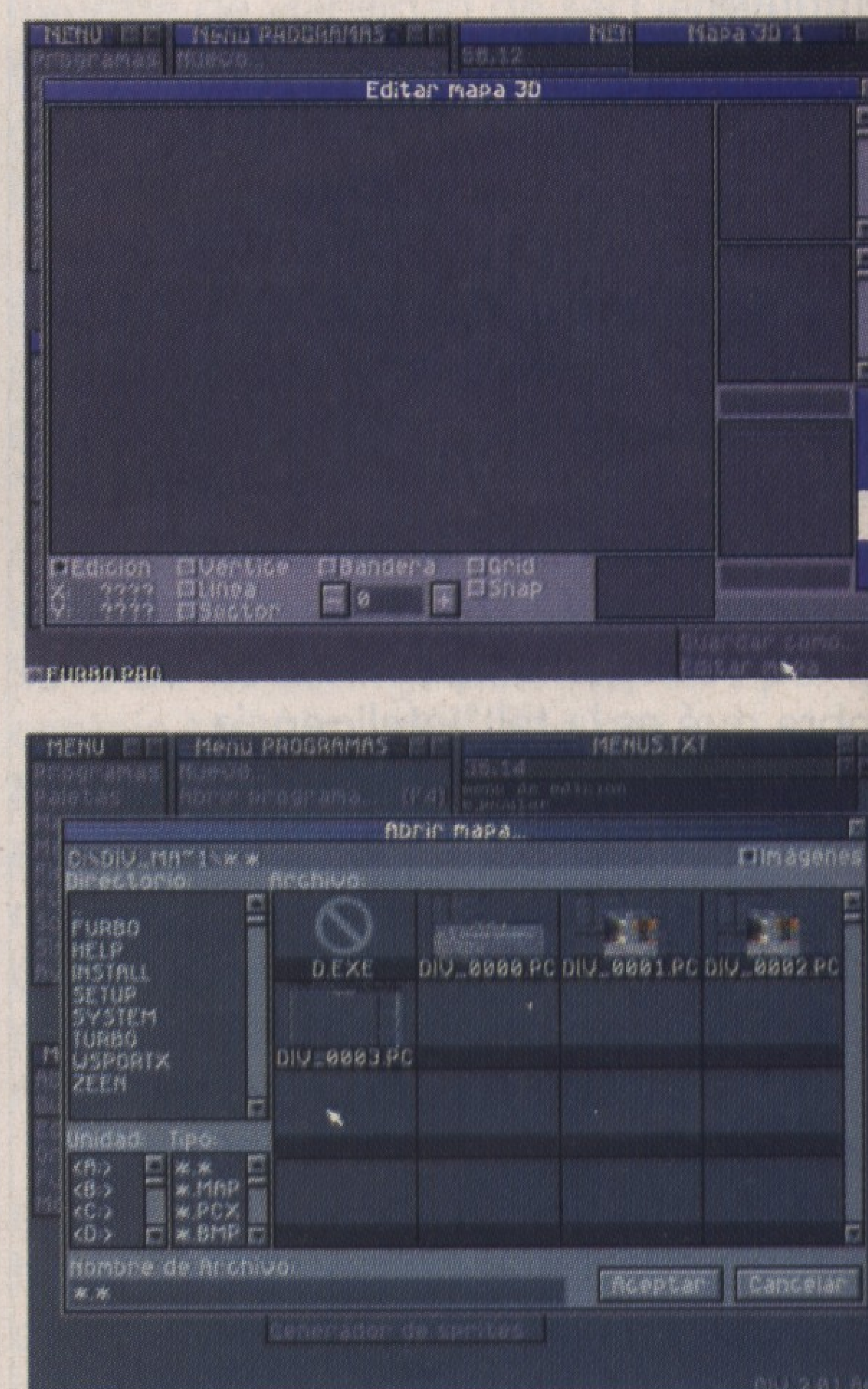
Esta función retorna el valor ASCII de la primera letra del literal que se le pasa como parámetro. Ejemplo *char*:

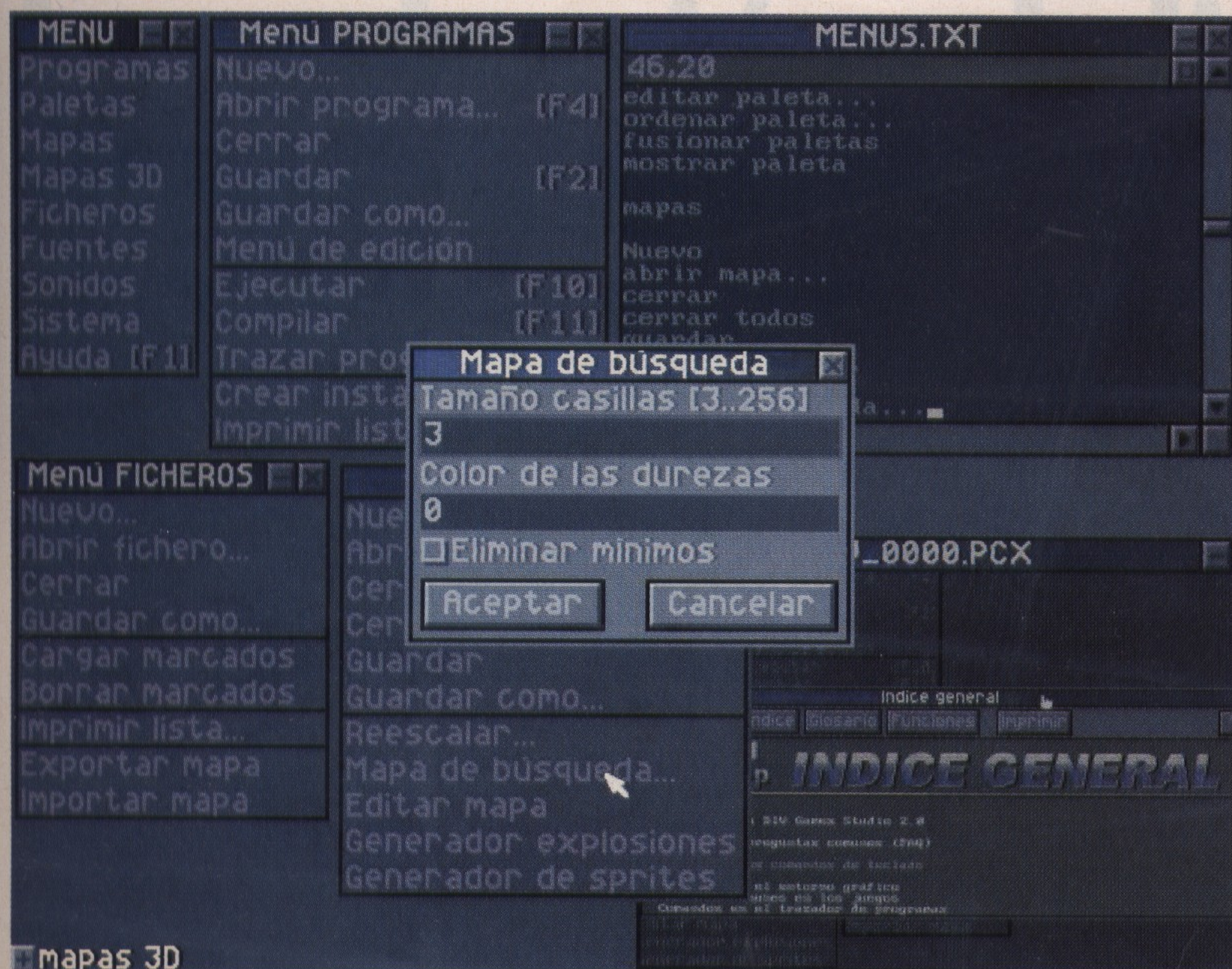
```
program ejemplo_char;
private
a,b;
begin
   a="a";        // a
   b=char("z"); // el valor de z
   es 122
   exit(a,b);
end
```

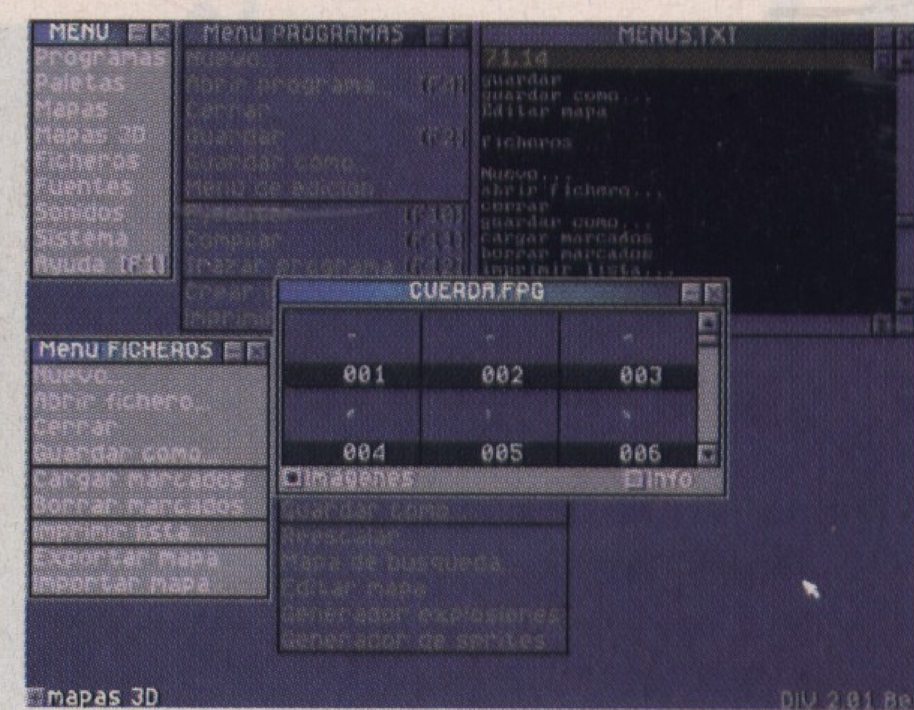
Otra es *upper (<cadena o carácter>)*. Convierte la cadena o el carácter a mayúscula. Esta función actúa de manera distinta según el parámetro que se le pasa. Si es una cadena se cambiarán de minúscula a mayúscula todas las letras de la cadena. En cambio, si es un único carácter no se cambia nada, sino que la función devuelve el código ascii de la letra.

Lower (<cadena o carácter>). Convierte la cadena o el carácter a











minúscula. Esta función actúa análogamente a lower según los parámetros que reciba. El ejemplo que se muestra a continuación convierte los caracteres que están en minúscula a mayúscula y viceversa:

```
program upper_lower;
private
    string a[18]="HoLa QuE tAl
    EsTaS?";
    string b[18];
    int i;
begin
    for(i=0;i<=18;i++)
        if(a[i]==lower(a[i]))
            b[i]=upper(a[i]);
        else
            b[i]=lower(a[i]);
        end
    end
    exit(b,0);
end
```

Strlen (<string o literal>), devuelve la longitud de un *string*. El ejemplo muestra el uso de strlen para conocer la longitud de la cadena, y así alterar el contenido de la misma.

```
program strlenght;
private
    string a[9]="HoLa";
    int len,i;
begin
    len=strlen(a);
    for(i=0;i<len;i++)
        a[i]="0";
    end
    exit(a,len);
end
```

Strset (<cadena>, <carácter>). Pone todas las posiciones de una cadena al carácter especificado. El carácter puede ser un literal o bien el valor ascii del mismo. El uso de esta función puede ser sustituido por un bucle:

```
strset(a,"")
```

Sería lo mismo que:

```
for(i=0;i<strlen(a);i++)
    a[i]="";
end
```

En el siguiente ejemplo se asigna a todas las posiciones de una cadena un determinado carácter mediante el uso de las dos maneras explicadas.

```
program ej_strset;
private
    string a[3]="Hola",b[4]="Adios";
    int i;
begin
    for(i=0;i<strlen(a);i++)
        a[i]="0";
    end
    strset(b,"1");
    exit(a+" "+b,0);
end
```

Strdel (<cadena>,<caracteres inicio>,<caracteres final>). Esta función elimina caracteres del inicio o del final de una cadena. Borrará tantos caracteres del principio como indique el segundo parámetro pasado a la función y tantos caracteres del final como marque el tercer parámetro.

Podemos borrar caracteres del final de una cadena usando el operador de sustracción (-), por lo tanto, escribir *strdel(a,0,1)* sería lo mismo que a--.

El siguiente ejemplo muestra como borrar caracteres de una cadena con los dos métodos explicados:

```
program ej_strdel;
private
    string a="Ejemplo de
    strdel",b,c;
begin
    strcpy(b,a);
    strcpy(c,a);
    strdel(b,rand(0,5),rand(0,5));
    c-=rand(0,10);
    exit("Cadena inicial: " + a + "
    ||| Cadena final(1er m,todo):
    " +b + " ||| Cadena final(2º
    m,todo): " + c,0);
end
```

Strcat (<cadena destino>, <cadena origen>). Une la cadena destino con la cadena origen, quedando el resultado en la cadena destino. Podemos reemplazar *strcat* mediante el uso del operador +, de manera que *strcat (a,"b")* sería igual que escribir *a+="b"*. Esto sólo puede realizarse si la cadena no excede los 1024 caracteres.

El siguiente ejemplo compone una cadena mediante el uso de *strcat*.

```
program strcat_ejemplo;
private
    string a[]="5b=";
```







```
    int i;
    byte est;
begin
    for(i=0;i<9;i++)
        if(!est)
            strcat(a,"b");
            //a+="b";
            est=1;
        else
            strcat(a,"+");
            //a+="+";
            est=0;
        end
    end
    exit(a,0);
end
```

Strcpy (<cadena destino>, <cadena origen>). Copia la cadena origen sobre la cadena destino, eliminando todo los datos que ésta contenía antes. La cadena destino tiene que ser forzosamente un *string* y la cadena origen puede ser tanto un *string* como un *literal*. Si la cadena es menor de 1024 caracteres se puede usar, en lugar de *strcpy*, una simple asignación (=). Según esto sería lo mismo *strcpy(a,"b")* que *a="b"*.

Strcmp (<cadena1>,<cadena2>). Compara dos cadenas y devuelve:
0: si las dos cadenas son iguales.
-1: si la primera cadena es menor que la segunda.
1: si la segunda cadena es menor que la primera.

De la misma manera que ocurre con otras funciones, *strcmp* puede ser reemplazado, siempre y cuando no exceda los 1024 caracteres, por los operadores de comparación (==,>,>=,<,<=,<>). Escribir *strcmp(a,b)* sería lo mismo que *if(a==b)*.

El ejemplo muestra el uso de las funciones *strcpy* y *strcmp*.

```
program str_cpy_cmp;
private
    string a="HOLA",b="";
begin
    strcpy(a,b);
    exit("Son iguales? (0=IGUALES , -1=A MENOR QUE B , 1=A MAYOR QUE B",strcmp(a,b));
end
```

Strchr (<cadena>,<caracteres>). Busca uno o varios caracteres dentro de una cadena. Esta función devuelve:
-1: si no se encuentra ninguno de los caracteres especificados.
N: N indica la posición donde se encuentra el primero de los caracteres especificados.

Esta función tiene un pequeño inconveniente: sólo busca hasta el primer carácter. Para solucionar esto la podemos reemplazar por un bucle. El siguiente ejemplo muestra cómo se puede hacer esto:

```
program ej_strchr;
private
    string a="Érase una vez un...";
    int pos[5],i,poscount;
begin
    for(i=0;i<strlen(a);i++)
        if(a[i]=="e")
            pos[poscount]=i;
            poscount++;
        end
    end
    write(0,0,0,0,"La e se encuentra en las siguientes posiciones:");
    for(i=0;i<poscount;i++)
        write_int(0,0,i*10+10,0,&pos[i]);
    end
    loop
        frame;
    end
end
```

strstr (<cadena>,<subcadena>). Esta función busca una subcadena en el interior de otra cadena. La función devuelve:
-1: Si no se encuentra la subcadena.
N: N es la posición donde empieza la subcadena.

En este artículo hemos tratado todas las funciones para la manipulación de cadenas de texto, y en el siguiente, complementando a éste veremos las funciones de lectura/escritura que incorpora DIV2.

Ramón de España (MaQNaZiLLeR)
http://maqnaziller.pagina.de
maqnaziller@pagina.de







Introducción al diseño gráfico

Cómo elaborar un guión

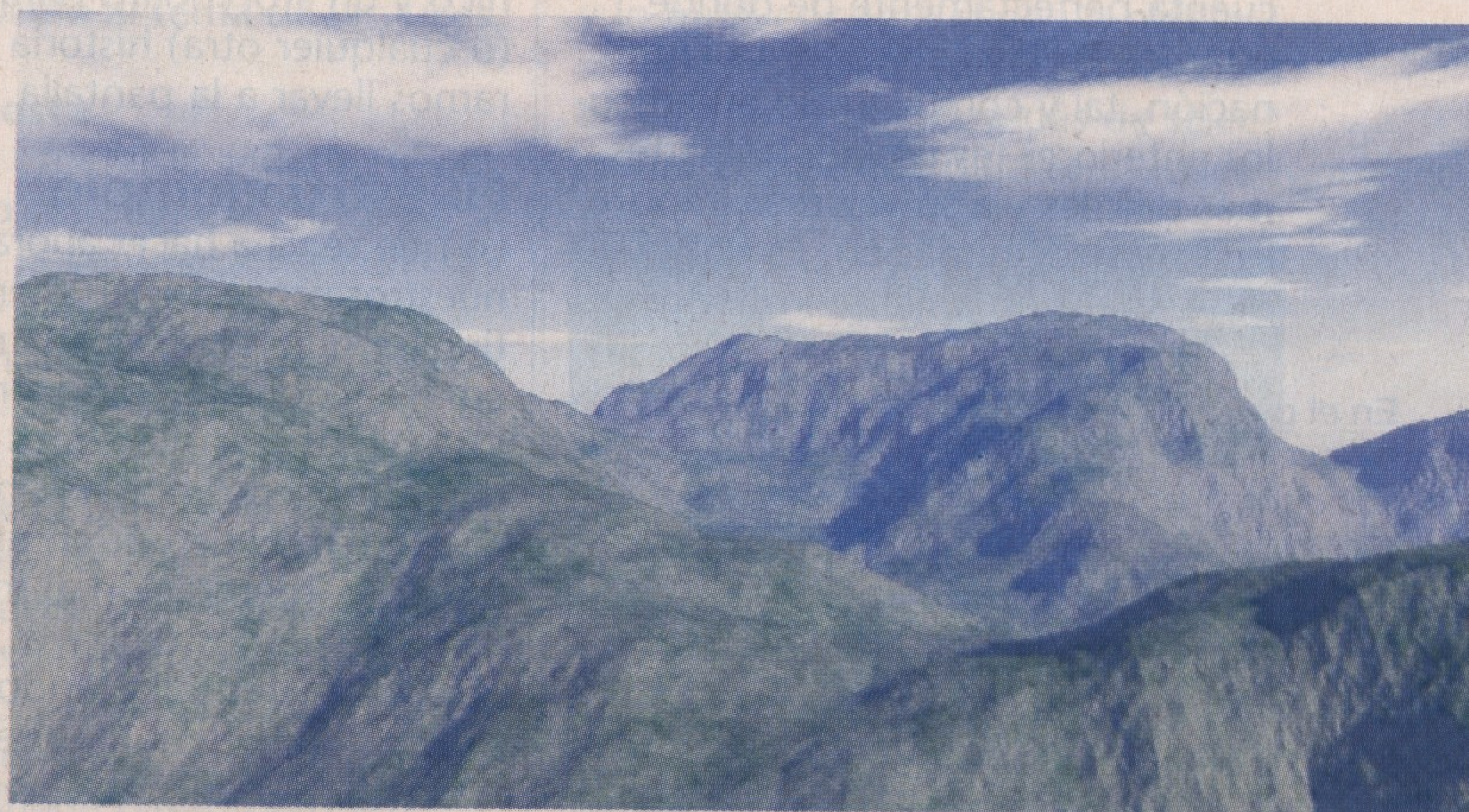
Si se hace memoria, en el artículo anterior vimos la definición de lo que podía denominarse un guión técnico, un storyboard, o una sinopsis argumental. En el presente artículo lo que se hará es profundizar en estos guiones para saber cómo elaborarlos y qué nos hace falta para realizar nuestros propios trabajos.

Los guiones son indispensables para hacer cualquier animación, anuncio publicitario o película. Vamos a ver los puntos más importantes a tener en cuenta a la hora de ponerse frente a la hoja en blanco.

Sinopsis argumental

Lo primero que se debe hacer es pensar en una historia que queramos llevar a la pantalla. Después de tenerlo más o menos claro, lo que deberemos hacer es escribir una sinopsis argumental. Se trata de llevar al papel los pensamientos sin especificar demasiado en la iluminación, los planos, ni los ángulos, simplemente contando la historia. No es bueno incluir demasiados detalles en este texto puesto que estos serán expuestos en el guión técnico que, como dice su nombre, está orientado más a los aspectos específicos. Esto no significa que en la sinopsis argumental tengamos que escribir algo tan simple como un telegrama. Sólo significa que tenemos que poner lo suficiente para entendernos a nosotros mismos y que cualquier persona que lea este guión sepa más o menos qué va a pasar. En definitiva, la sinopsis argumental es la explicación escrita del argumento de nuestra historia.

La sinopsis argumental es la explicación escrita del argumento de nuestra historia



Guión técnico

Seguidamente se debe proceder a crear el guión técnico, que se podría decir que es una versión más especializada de la sinopsis argumental. En el guión técnico se deben especificar un montón de detalles que queremos en la representación audiovisual, así como la música, los ángulos, la localización, la iluminación, etc. Exactamente, los pasos a seguir y las cosas que se deben incluir son las siguientes.

Pasos

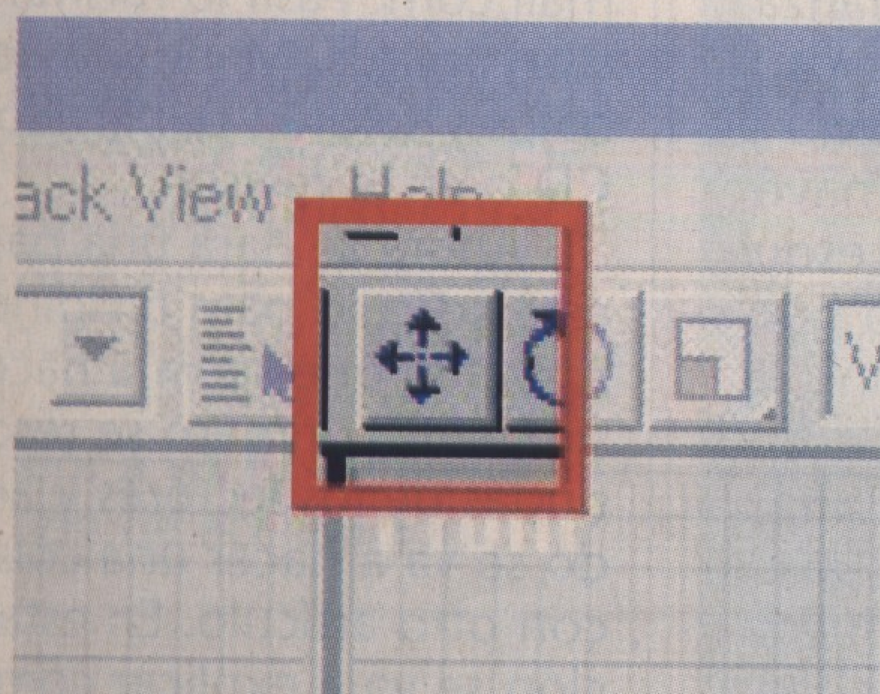
Cada hoja que se utilice para el guión técnico pertenecerá a una secuencia que es un conjunto de planos o un plano solo. Cada secuencia (hoja) debe tener los siguientes aspectos:

Primeramente, se debe especificar dónde se grabará o dónde sucederán los hechos.

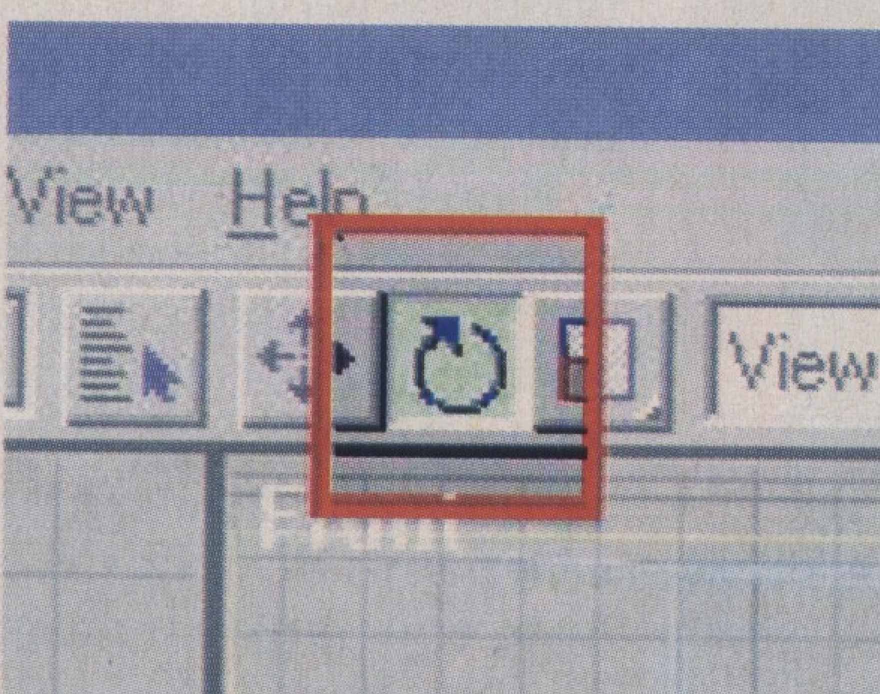
Seguidamente, se debe decir a qué hora del día (en la animación no se tiene muy en cuenta).

Y a su vez, cada secuencia u hoja está dividida en planos, de manera que se dividirá la hoja dependiendo del número de planos que tenga la secuencia. Cada plano debe tener algunos apartados que es conveniente poner:

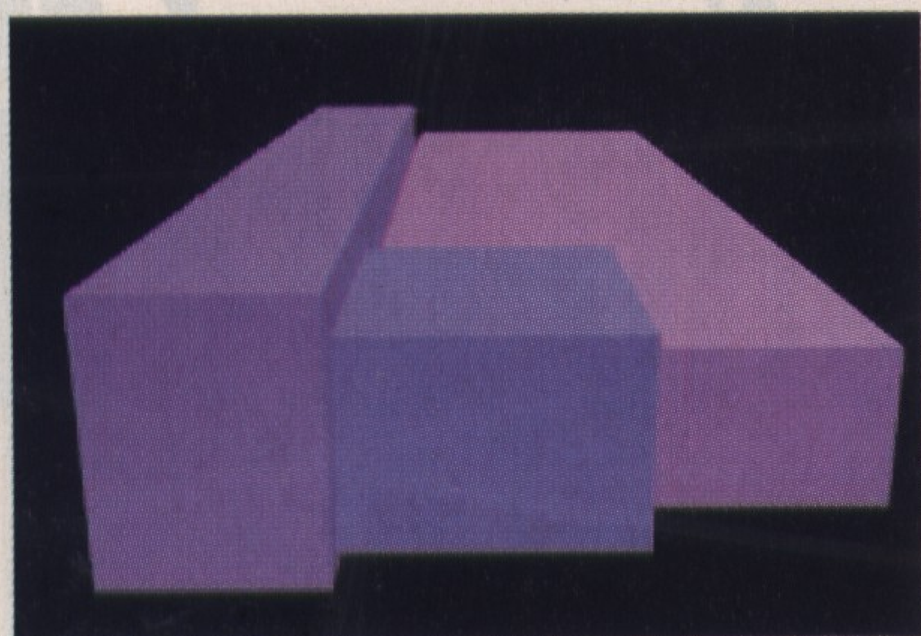
- Nº plano
- Iluminación
- Ángulo
- Música
- Otros



El botón para mover objetos.



El botón para rotar objetos.


La cámara sin rotar.


La cámara rotada.

En estos apartados se debe decir cuál es el número de plano dentro de una misma secuencia, de manera que podamos llevar la cuenta perfectamente de dónde estamos en la animación. La iluminación, tal y como se dijo en artículos anteriores, es una de las partes más difíciles y a su vez más importantes de la animación. Es por eso que es tan sumamente importante especificar cómo es la luz, de dónde viene, o si es dura o difusa. Cuantos más datos de luz demos más fácil y definida tendremos la animación. De no ser así, el grafista puede encontrarse con problemas y no ponerse de acuerdo con el que ha realizado el guión.

En el guión técnico se debe especificar en detalle lo que queremos hacer y cómo vamos a realizarlo

Storyboard

Una vez realizado el guión técnico se debe pasar a hacer el StoryBoard. Éste es una serie de dibujos que explican la historia a través de viñetas con un aspecto más visual que el texto. Se puede acompañar con algo de texto para especificar la acción pero no es completamente necesario.


Ejemplo de modelo poligonal en 3D.

Así pues, se debe hacer una sinopsis argumental, un guión técnico y un storyboard para nuestra (o cualquier otra) historia que queramos llevar a la pantalla.

Nuestro ejemplo

Lo que se va a hacer ahora es llevar nuestros proyectos en mente a este tipo de guión antes de la realización en 3D, ya que ahora tenemos todos los objetos pero debemos pensar como moverlos por la pantalla.

Podemos pensar en una animación de presentación al más puro estilo de juegos de naves, donde se ve a los malos aplastando a los buenos o al revés. En nuestro caso nos vamos a decantar porque los malos aplasten a los buenos y de esta manera al jugador le entren más ganas de venganza.

Nuestra sinopsis argumental

Para empezar se debe realizar una sinopsis argumental. En nuestro caso será la siguiente:
A lo lejos se puede observar un planeta que parece de lo más tranquilo, de repente sale una nave que parece estar en apuros. En ella podemos ver el sello de la alianza. Tras ella parece haber dos naves dispuestas a acabar con el navío de la alianza a base de láser. La nave de la alianza parece intentar esquivar los láser pero cae abatida por los "malvados". Tras una gran explosión se puede observar como las dos naves avanzan hacia la cámara (es decir nosotros) y pasan. Justo al pasar la pantalla se queda de color negro total y se ven unas letras: "80x86 WAR".



Al parecer, hasta ahora, todo es muy sencillo de realizar, pero ahora viene una de las partes más complejas: hacer el guión técnico. Para su realización se debe tener muy claro lo que se va a hacer y cómo se va a hacer.

Las partes de suma importancia en esta animación son la iluminación del planeta y de las naves. El autor cree que para esta animación en especial la iluminación del planeta y de las naves debería ser lateral, de manera que poco trozo del planeta quedará iluminado, igual que las naves; aunque éstas estarán algo más iluminadas puesto que están a una cierta distancia del planeta y su tamaño es mucho menor.

Como cada persona haría un guión técnico diferente le vamos a dejar aparte y se deja libertad para que cada lector haga el suyo propio, de manera que cada uno añada los detalles que más le gusten, eso sí, sin salirse de los márgenes de la sinopsis argumental.

Dentro de dos números de la revista se publicarán algunos de los guiones técnicos de los lectores que se ofrezcan voluntarios a mandar los que consigan hacer a la siguiente dirección: thorse3d@hotmail.com. Pasa lo mismo con los storyboards. Mandad vuestros trabajos y saldrán publicados en el CD o en la revista.

Ahora se hará una pausa en este artículo para dividirlo en dos. Ahora lo que se va a hacer es lo siguiente: primero se resolverán las dudas de los lectores y acto seguido se va a hacer una colaboración con otro artículo. En esta colaboración se va a explicar cómo desarrollar un personaje.



Las texturas son estupendas en los personajes de Quake III.

Dudas de los lectores

Hemos recibido cuatro dudas durante estos dos últimos meses. Esto es bueno, la gente empieza a expresar sus dudas y los que lo han hecho ya saben que el autor de este artículo no muerde, que responde a los mails y pide permiso a los lectores para publicar sus dudas. Esta vez, los 6 mensajes han sido enviados por Alberto Fuentes (Zyope).

1ª Duda

Cuando instalo el *3d Studio Max* me dice que seleccione entre HEIDI, OPENGL y DIRECT3d. ¿Cuál es mejor?. Yo uso HEIDI. Tengo una Matrox Millenium G200. Gracias

1ª respuesta

Esta es una pregunta con la que nos hemos encontrado todos nosotros al iniciar *3DS MAX* por primera vez. Es algo que confunde un poco. Por hacerlo sencillo podríamos decir que el programa quiere saber qué aceleración ha de utilizar para funcionar más rápido. Esto es porque la gran cantidad de tarjetas que actualmente hay en el mercado aceleran algún tipo de tecnología, como OPENGL o la de DIRECT3D, pero para saber si nuestra tarjeta tiene soporte HEIDI lo mejor es mirar los manuales de la misma, si no dice nada, es poco probable que exista algo para tu tarjeta. No obstante puedes buscar en el WEB SITE de tu tarjeta, en tu caso creo que es www.matrox.com, y buscar entre los drivers y soporte al consumidor a ver si tu tarjeta tiene este tipo de aceleración.

2ª Duda

Cuando ilumino la copa sobre la mesa o los ovnis sobre un campo de trigo (artículo de Divmanía 3), la luz no ilumina el suelo, o lo ilumina muy mal y no sé por qué. Creo que podrías dedicar un artículo al tema de las luces. Saludos.

2ª Respuesta

Era de esperar la llegada de un mail como éste. En efecto, como ya se dijo, la iluminación es un tema muy complejo y se le va a dedicar un artículo entero, pero de momento podemos pasar sin hacerlo, o hacerlo como sabemos. Respecto a tu fallo en la iluminación pueden ser 2 cosas. La primera puede ser un fallo en *MAX*, cosa que no me extrañaría demasiado y la segunda opción es que realmente no te salga, que también puede pasar, pero yo confío en que lo estás haciendo bien.



3ª Duda

Hola, soy Zyope, escribo por el artículo de Divmania nº 4. El funcionamiento del *EditMesh* para transformar un cubo no quedó bastante claro. De la forma en que yo lo hago puedo sacar cubos, pero no meterlos. Además es muy difícil mover partes en concreto. Yo creo que lo estoy haciendo mal. ¿Podría explicarse mejor? Gracias anticipadas.

3ª Respuesta

Hola Zyope, los problemas con el EditMesh son muy frecuentes, aunque hay algún que otro truco para solucionarlos. Puedes, en vez de ir modificando todo el rato las caras, tocar los vértices, para dejarlos en una posición tal que creen una cara en una orientación difícil de conseguir solamente modificando las caras. Para hacer esto, en vez de ponerte en el apartado "faces" te pones en "vertex". Seleccionas el vértice deseado y lo mueves como si de un objeto se tratara. Otra cosa a tener muy en cuenta es que después de seleccionar la cara que quieres, puedes

Algunas dudas de los lectores servirán para esclarecer puntos que han quedado poco claros



Animar este tipo de modelos en 2D cuesta bastante más trabajo.



Las facciones de los personajes de las aventuras gráficas parecen tener vida.

usar un modificador llamado "face extrude". Con él es posible estar jugando con el tamaño y el espacio donde está la nueva cara que estamos modificando. Si no puedes meter caras hacia dentro creo que se debe a que tienes que duplicar la cara otra vez. Es decir tenemos una cara, ésta la haces más pequeña. Una vez disminuida la duplicas y, sin moverla, la haces más pequeña. Una vez realizado este paso, vuelves a duplicarla y de esta manera la puedes meter o sacar del objeto. Espero que te haya quedado claro aunque la explicación es algo confusa.

El modelado de la cabeza es la parte más difícil a la hora de crear un personaje poligonal

4ª Duda

Leyendo el artículo me han asaltado algunas dudas: ¿de qué sirve el modificador UVW MAP que le aplicas al meteorito? También dices en el artículo que, en el número anterior, enseñabas a hacer cámaras, pero no lo he visto por ningún lado.

4ª Respuesta

El modificador de UVW MAP es un modificador que afecta a las texturas, más bien es de la forma que se aplican las texturas. Para poner un ejemplo sencillo imaginemos por un momento que tenemos un objeto delante nuestro. Este objeto es un cubo. Seguidamente nosotros queremos hacer que tenga aspecto de trapo a cuadrados. Si cogemos una tela con esa textura podemos ponerla sobre el objeto de varias maneras. La más fácil sería de forma plana, que es simplemente poner encima del cubo el trapo, pero debemos tener en cuenta que si tuviéramos el trapo estirado por todos lados (sin dejar que hiciera efecto la gravedad), podríamos observar entonces que las partes laterales del objeto no quedan con la tela orientada para que se vea correctamente; y además, la parte de abajo la tiene invertida, cosa que tampoco es buena. Para *mapear* correctamente un cubo, deberíamos utilizar el UVW MAP y ponerlo en modo cúbico, de manera que cada una de las caras tendría la tela orientada correctamente. En el caso de una pelota usaríamos el *mapeado* redondo, en el caso de una tubería lo más probable es que usáramos el *mapeado* cilíndrico. Dependiendo del caso.

Tu segunda duda de este *mail* era como crear una cámara. Pues a continuación lo explicaré.

Creando una cámara

El procedimiento a seguir para la creación de cámaras es sencillo. Lo que se debe hacer es acceder al panel "crear" y seguidamente darle a la opción "cameras". Una vez dentro podemos escoger entre dos tipos de cámara: las *target* y las *free*. Principalmente usaremos las *target*, que nos permitirán un mayor control en lo que queremos enfocar y cómo se quiere hacer. Acto seguido, para colocarla en el espacio 3D lo que se debe hacer es pulsar una vez sobre el espacio 3D, y sin dejar de pulsar ir a otra parte del espacio y entonces soltar el ratón. Lo que se hará con esto es crear una cámara. Cuando se ha hecho clic lo que se ha hecho es definir el origen de la cámara mientras que cuando la hemos soltado hemos definido el destino. El cuadro que queda en el destino es la zona que se verá con la cámara. Una vez creada se puede ir a la vista perspective y desde ahí pulsar la *tecla C* lo que hará que esta última vista pase a ser una vista de cámara mostrando lo que el objeto "cámara" es capaz de ver.



Modificando la posición de la cámara

Para modificar la posición de la cámara lo que se debe hacer es modificar su posición como con cualquier otro objeto, es decir utilizando el icono "mover". Este icono es como el de la figura 1.

El funcionamiento de este botón es bastante simple. Lo que se debe hacer para mover cualquier objeto es pulsar sobre él y, sin dejar de pulsar con el botón, arrastrar el objeto. Una vez en el sitio deseado se debe dejar de pulsar el botón.

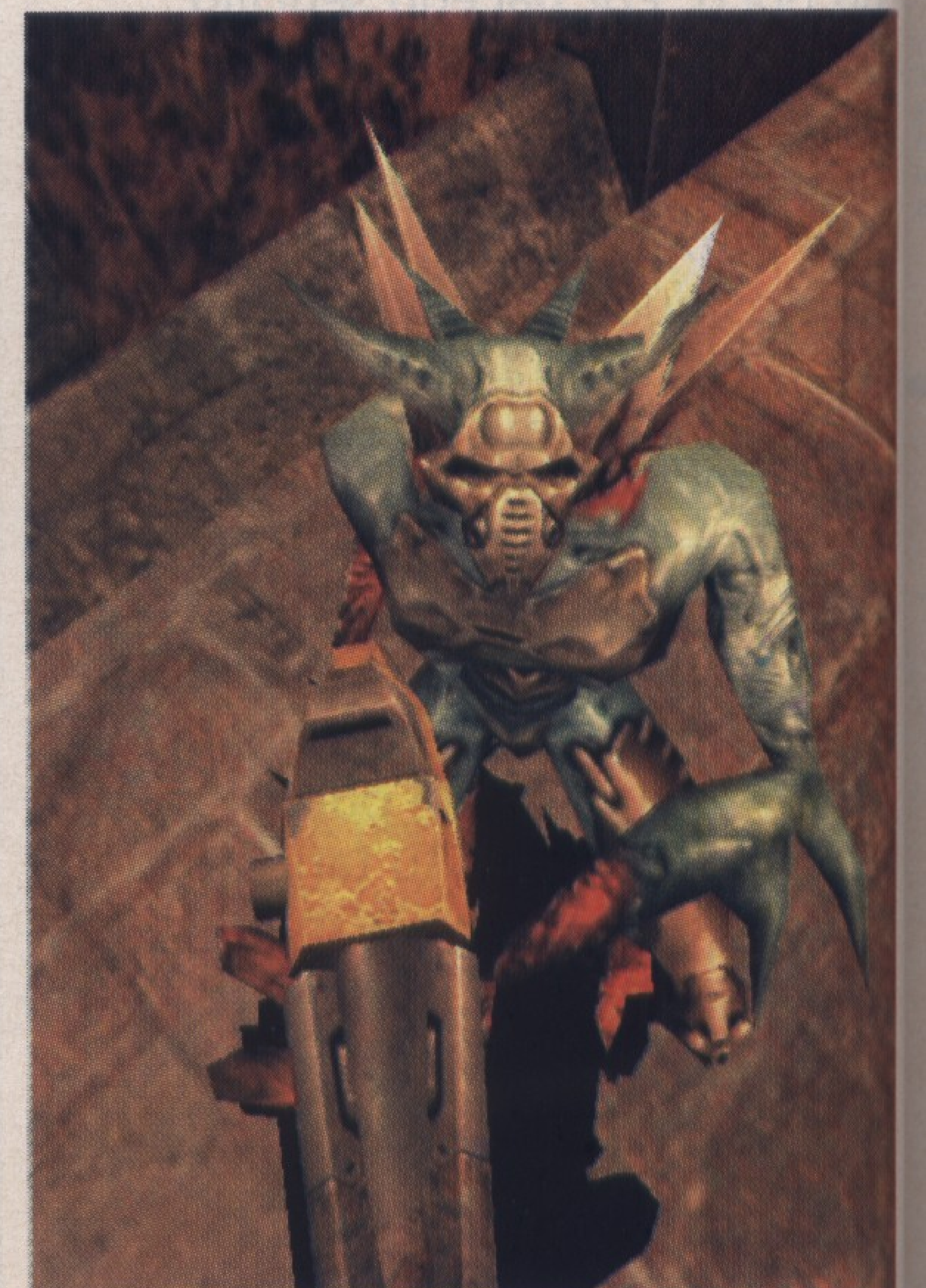
Rotación la cámara

Si se quieren hacer efectos, como que la cámara rote sobre sí misma o semejantes, lo que se debe hacer es lo siguiente: se debe seleccionar el origen de la cámara y pulsar sobre el botón de rotación que es el de la figura 2.

El procedimiento es el mismo que con la cámara, no se suelta el botón del ratón hasta conseguir el efecto deseado. Se puede observar el resultado de rotar la cámara en las figuras 3 y 4.

Realizando un personaje 3D

Ahora se va a proceder a la creación de un personaje tridimensional. Para crear cualquier personaje tridimensional lo que se debe tener siempre en cuenta es cómo se va a modelar, si poligonal o si por el contrario será orgánico. Ejemplos de modelado poligonal pueden ser los personajes de *Unreal* o *Quake*,



Una cara bastante trabajada.



que consiguen un aspecto bastante real gracias a las texturas. Otro tipo de personajes son los orgánicos.

Los personajes poligonales se suelen utilizar en juegos con un engine 3D en tiempo real, lo que significa que cada objeto es construido en ese momento y queda afectado por el resto de los objetos. Los juegos que utilizan este tipo de *engines* son los de tipo *Unreal* o *Quake* como hemos dicho; mientras que juegos tipo aventuras gráficas son títulos en que los gráficos no suelen ser poligonales. Alguna excepción podría ser el *Grim Fandango*, donde los gráficos son una mezcla de *engine 3D* y *2D*. Este es un nuevo método para crear aventuras gráficas ideado por Lucas Arts.

En este artículo hablaremos de la creación de un personaje poligonal. Quizás, dentro de algunos artículos, se trate del modelado de un personaje enteramente orgánico (se hará después puesto que es más difícil). Como todos sabemos actualmente, DIV, no soporta *engine 3D*. Bueno, soporta el modo8, pero no podemos poner objetos de *3ds MAX* en él. Quien sabe si con el tiempo, los creadores de DIV lo implementan.

Tanto el modelado poligonal como el orgánico llevan mucho trabajo y no sé cuál es más difícil. Todo depende de las ganas de trabajarse el modelo y para qué se vaya a utilizar.

Cómo modelarlo

Como ya se ha dicho anteriormente se va a modelar una forma poligonal, de manera que sea menos trabajoso, pero también obtendremos un resultado algo menos espectacular.

Para el modelado poligonal se suelen usar cosas como la edición de vértices y las figuras *boleanas*. Las figuras *boleanas* se basan en unir o crear nuevos objetos a partir de dos de estos. Por lo general, lo más usado es la edición de vértices. Una vez modelado el personaje en sí, será hora de asignarle una textura medianamente convincente. Para ello se utilizaran algunas fotos o bien dibujos.



Empezar a trabajar

Antes de empezar se debería tener alguna fotografía o dibujo de cómo debe ser nuestro personaje, para no equivocarse con las proporciones ni nada parecido.

Se podría empezar con crear un cubo en la parte del estomago e ir modificando vértice a vértice hasta conseguir la cintura y la parte de arriba del cuerpo sin la cabeza ni los brazos. Acto seguido se debería ir añadiendo vértices y segmentos usando el modificador *EditMesh*.

Una vez realizado podríamos seguir los mismos pasos para los brazos y luego para la cabeza (la parte más difícil del cuerpo entero). Acto seguido, se debería crear el modelo de cintura para abajo. Trabajar con las texturas y el modelado poligonal es de gran utilidad, ya que se pueden obtener grandes resultados gracias a una buena textura, aunque el modelo no nos acabe de convencer. Un ejemplo de ello podrían ser los personajes de *Unreal*, que tienen un aspecto bastante real y eso gracias a las texturas. Lo mismo ocurre con juegos como *Fifa* (actualmente en su versión 2000), que hace maravillas sólo con la textura de sus personajes. Eso sí, el modelo debe parecer mínimamente humano, si se parece a un árbol es que se va por mal camino.

Es conveniente no juntar las diferentes partes del cuerpo, porque a la hora de poner las texturas es mejor hacerlo por separado, al igual que si queremos mover el personaje. En tal caso, lo mejor es que éste esté dividido en diferentes partes que se puedan mover independientemente de cómo esté el resto del cuerpo, ya que en caso contrario deberíamos modificar vértice por vértice ¡incluso en las animaciones!

Pasos

Así pues los pasos y los trucos para poder modelar nuestro personaje poligonal correctamente son los siguientes:

Empezar desde alguna parte del cuerpo parecida a una forma geométrica e ir añadiendo vértices y segmentos para poder modificar mas cómodamente y obtener un resultado mejor.



Los personajes de dibujos animados se suelen modelar en 2D.

Trucos

No juntar nunca las partes diferenciadas del cuerpo, como los brazos ni la cabeza, para después poder animar de una forma más sencilla y agradable. Se debe recordar que si uno se pasa modificando y hace todo el modelo a partir de modificaciones de un cubo (que además es muy difícil), después tendrá dificultades para moverlo adecuadamente. Por eso no se recomienda esa técnica.

No se debe hacer todo el modelo a partir de modificaciones de un cubo o habrá dificultades para moverlo

Esto es todo por ahora. En el siguiente número se va a profundizar en la creación de este personaje que hemos empezado a hacer. Recordad los consejos anteriormente mencionados porque son muy importantes. Sólo decir que si tenéis dudas podéis escribir a thorse3d@hotmail.com. En efecto, la dirección de correo ha cambiado pero ha sido inevitable por problemas con el servidor donde antes el autor tenía el correo. Así que podéis mandar vuestras sugerencias, críticas, imágenes y comentarios a la dirección anteriormente mencionada. ¡Hasta el próximo artículo!

David Martínez (thorse3d@hotmail.com)



Unreal Tournament es otro ejemplo perfecto de juego con modelos poligonales.

Vital Web

Revista electrónica sobre DIV

En el número cuatro de DIVmanía recogíamos la noticia de la puesta en marcha de una nueva revista electrónica, la publicación on-line VitalWeb. Ya es hora de que demos un repaso a sus jugosos contenidos. También mencionamos unos cuantos canales de chat dedicados al mundo de la programación en DIV.

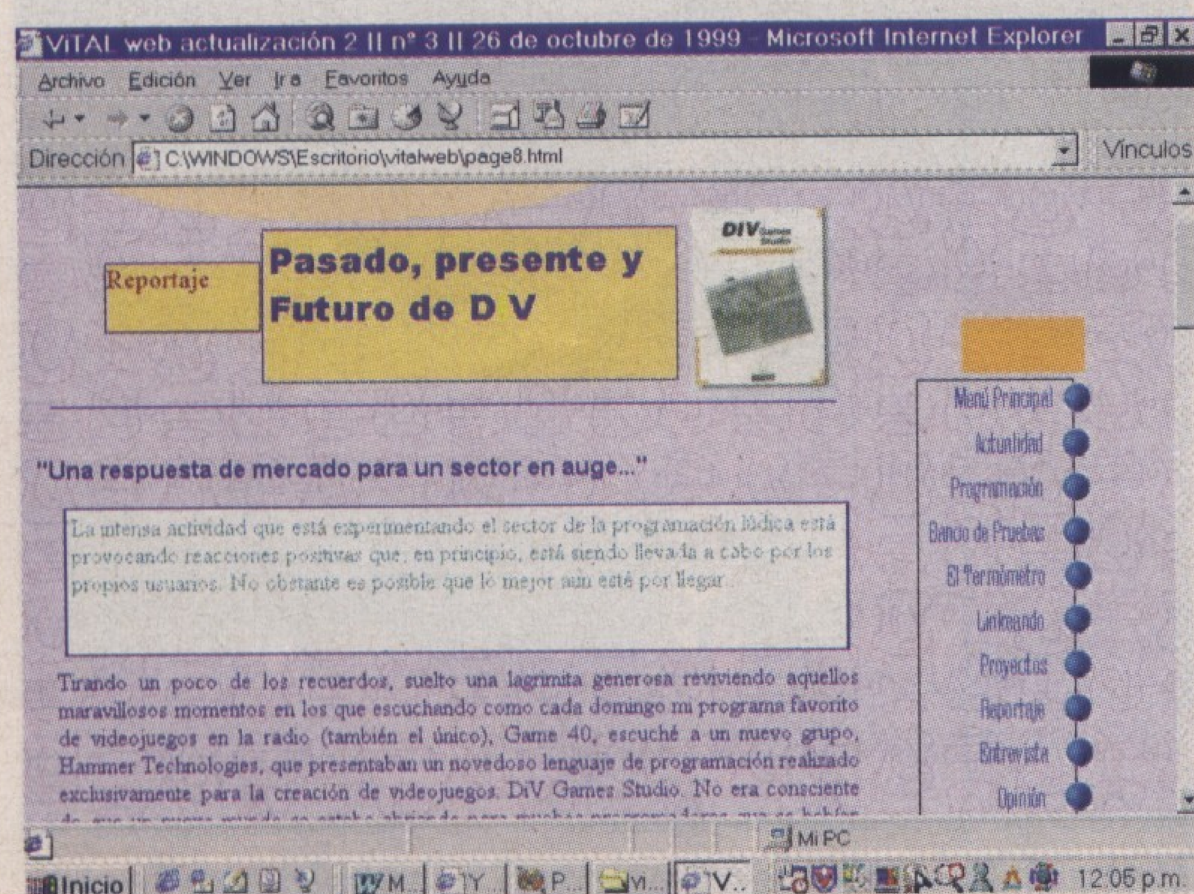
Las revistas electrónicas que podemos encontrar en Internet empiezan a ser tan abundantes que, haciendo un símil micológico, cualquier avezado buscador podrá llenar su canastillo de estas sabrosos setas electrónicas. Y es que crecen como los hongos después de una lluvia otoñal.

Y DIV no podía ser una excepción a este florecimiento. Si hace unos números hablábamos de DIVnet, una página dedicada en exclusiva a recoger la más rabiosa actualidad sobre el mundo de la programación de videojuegos, ahora toca hablar de VitalWeb.

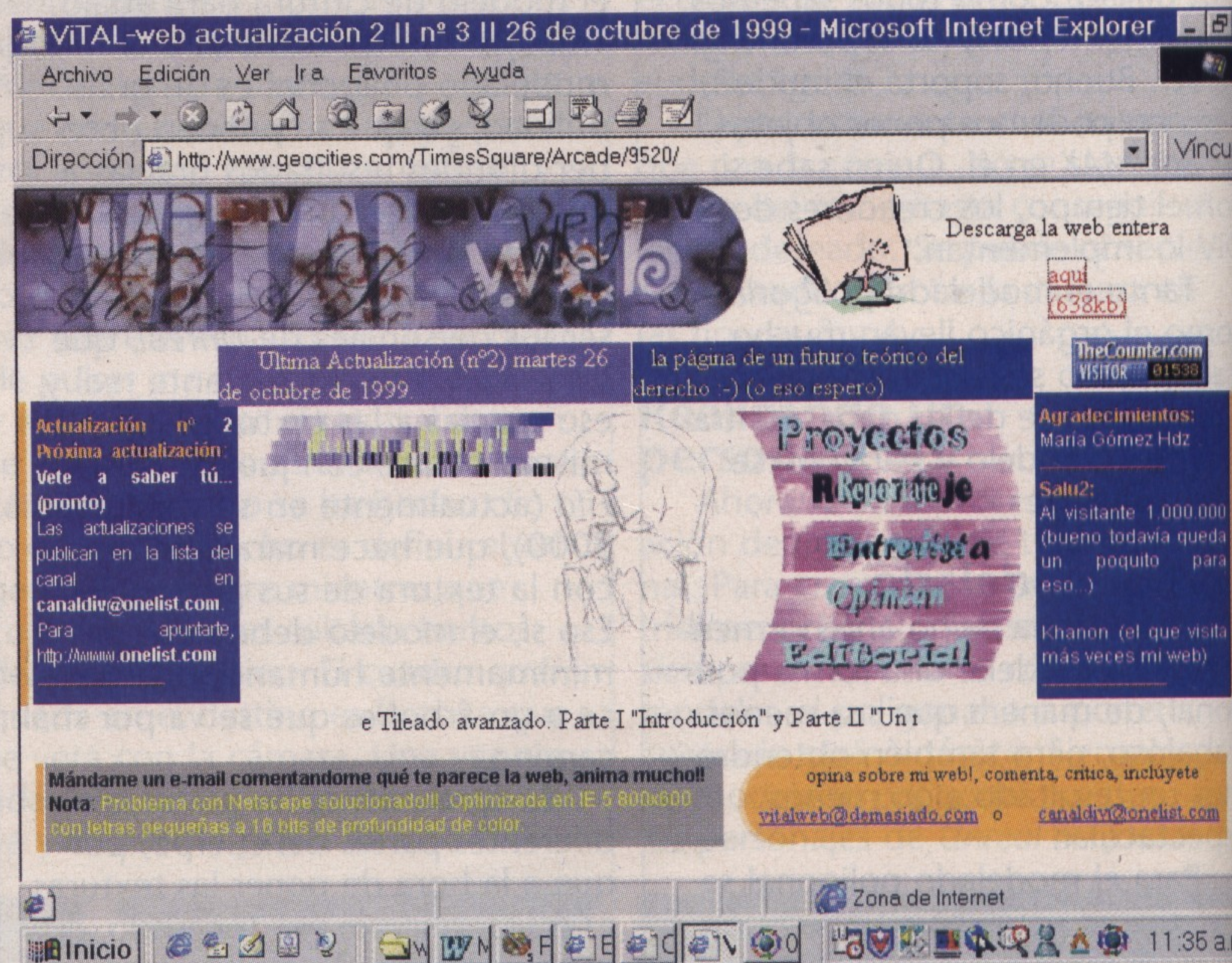
Otra revista electrónica enfocada a informar a todos los interesados en la programación de juegos

Vital y completa

Lo primero que nos encontramos al cargar la dirección de VitalWeb en nuestro explorador es una pantalla con los contenidos de esta página, que vamos a diseccionar seguidamente:



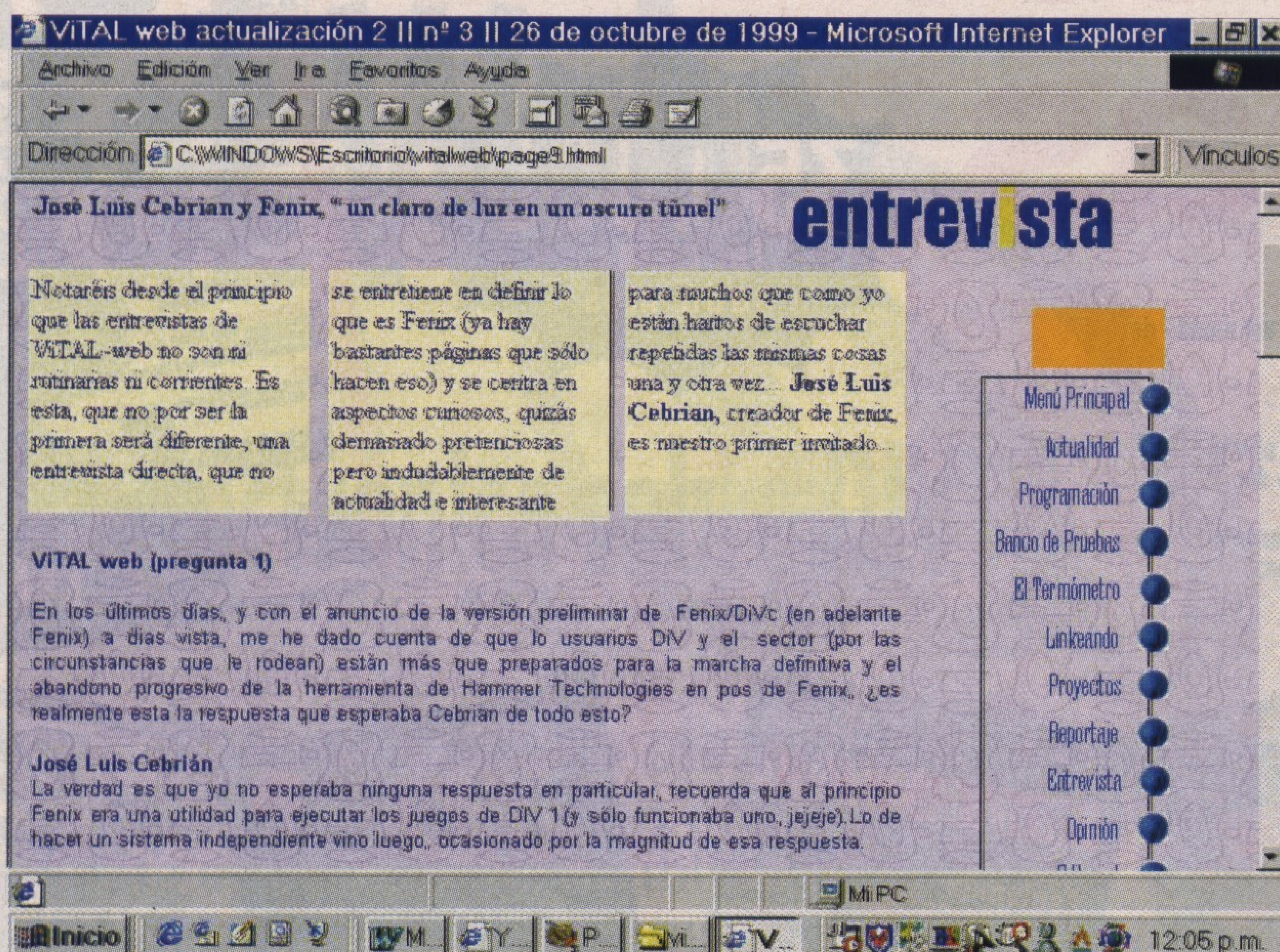
Un interesante reportaje sobre DIV.



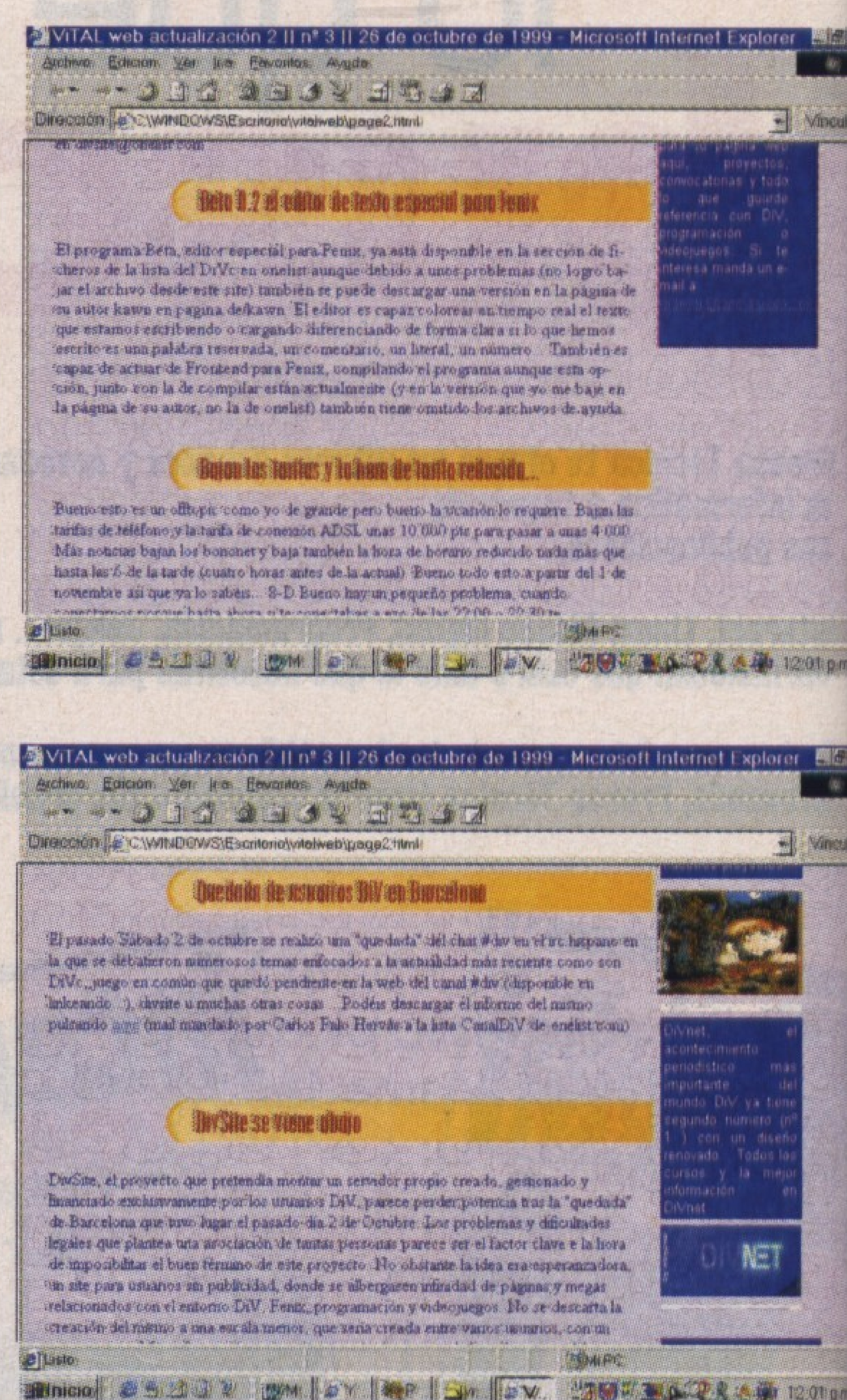
En la sección de actualidad se anuncia la intención de reescribir y reestructurar el código del nuevo compilador para lenguaje DIV, *Fenix*, y que el creador de la página se ha unido al grupo Papada Soft, un activo grupo de programación ya visitado en esta sección, para gestar un nuevo juego del género aventura/rol. También se anuncia que el número 1 de la revista hermana, DivNet, ya está disponible en la Red. Se puede ver un informe sobre los temas tratados en la última "quedada" del grupo de usuarios del canal de chat dedicado a DIV en el irc.hispano. Se comentan los problemas por los que ha pasado DivSite, un proyecto de crear un servidor propio gestionado y financiado exclusivamente por los usuarios de DIV. Se anuncia la disponibilidad de la versión beta de un editor de texto especial para *Fenix*. Y, por último, un comentario a la última bajada de tarifas de telefónica.

En el apartado dedicado a la programación con DIV se anuncian varios cursos. El primero de ellos está enfocado a conseguir un dominio total de los mapeados tiles. Los próximos cursos que estarán a disposición de los visitantes de esta página serán los siguientes:

Trucos generales en programación y DIV.



Entrevista con el creador de Fenix.



Uso de mapas de dureza y consejos en programación de rpg y estrategia.

También hay una sección de consulta de los lectores, para que envíen dudas o sugerencias sobre estos cursos.

Pasamos página. En el banco de pruebas se analiza el código de algunos juegos que se distribuyen a través de Internet o por medio de la revista que tienes entre tus manos. Se trata de estudiar estos códigos, para ver cómo se pueden mejorar, o alabar los aspectos acertados de la programación de los mismos.

VitalWeb es una revista con contenidos la mar de interesantes, entre ellos reportajes y entrevistas

En esta ocasión los juegos elegidos para analizar han sido *Master of Elements* de Nightwolf y *Tokenkai* de Hammer Technologies. Se ofrece una crítica, siempre constructiva, de ambos títulos y se muestran algunas capturas de ambos.

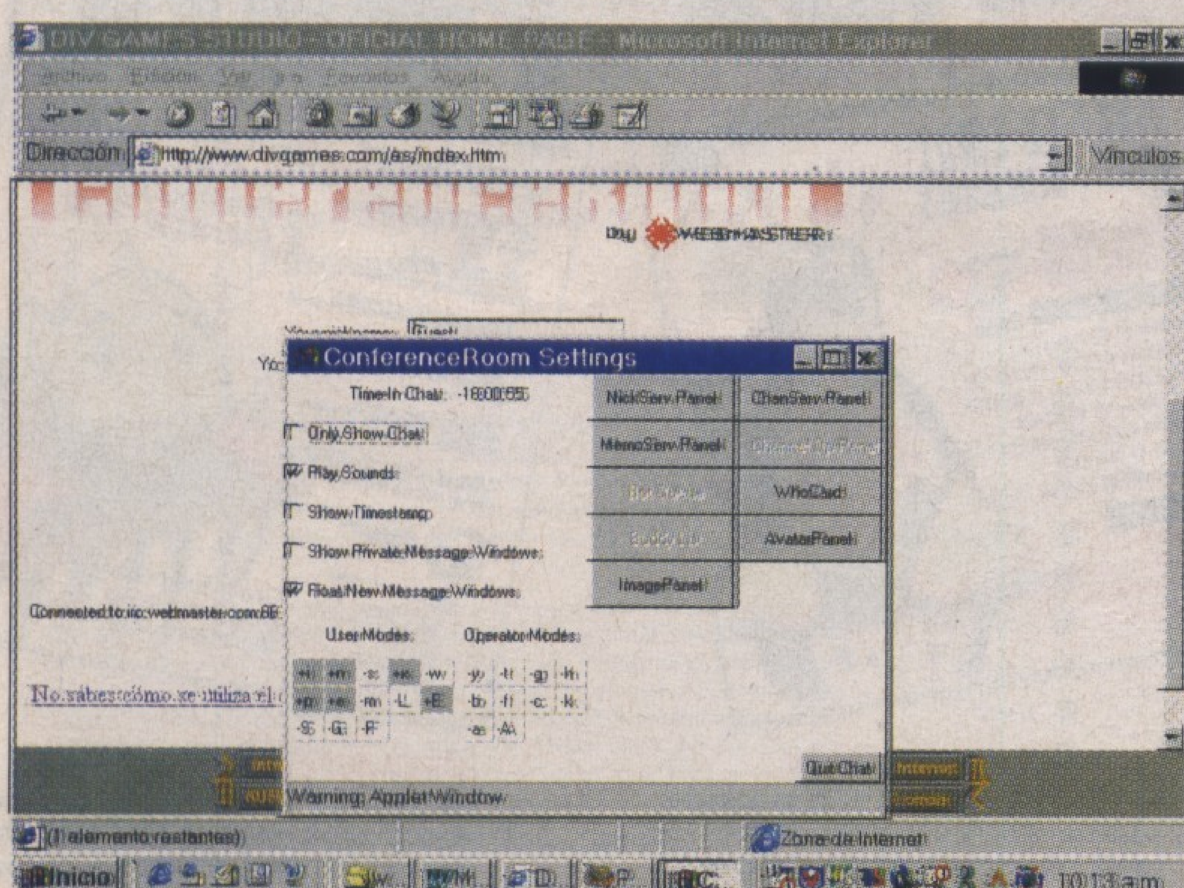
En "el termómetro" se pretende elaborar una lista de los mejores y los peores juegos programados con DIV. Como bien se avisa, no se pretende menospreciar a nadie con estas clasificaciones, sólo se busca el promover una sana competencia entre los diferentes grupos de desarrollo existentes en nuestro país. Entre los mejores juegos se encuentran *Master of Elements, ShowDown, Disco-Fighter, La Bolsa, Pablo Copter, Zelda* y *Maqdiablo*. En la lista de peores juegos no hay colocado ningún título, parece que se tiende a la benevolencia en VitalWeb.

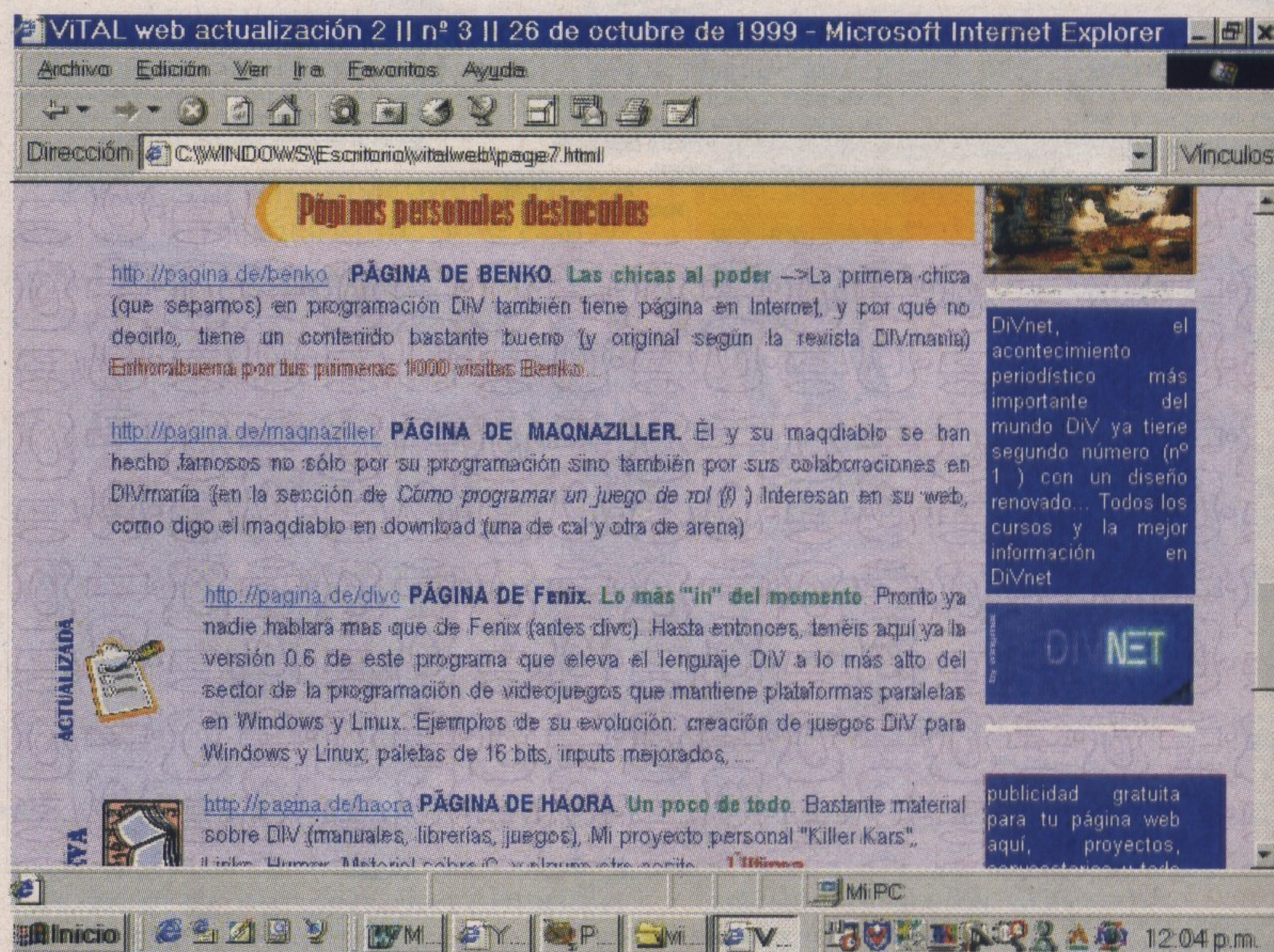
Proyectos: primero realizar un programa especial para realizar mapeados tiles; segundo empezar a programar un juego de rol y arcade tipo *Zelda*, pero ambientado en una mezcla de escenarios medievales y espaciales. Aquí tenéis el argumento de este nuevo proyecto:

"En una época medieval se están sucediendo hechos muy extraños que nada parecen tener relación con la realidad. Hombres del futuro se mezclan con los del medievo en una lucha por una planta, extinta en el futuro, con la que se elabora un combustible imprescindible para la buena marcha de un mundo cibernético. El protagonista deberá, tras salir de su asombro por las magias tan virtuosas de sus enemigos, combatir a las poderosas fuerzas venidas del futuro supliendo la falta de tecnología con buenas dotes de magia, inteligencia y sarcasmo anacrónico".

Si estás interesado en unirte a este proyecto y eres grafista 2D o 3D, guionista o programador familiarizado con el entorno DIV, parece que en VitalWeb necesitan gente que colabore con ellos.

También hay un reportaje dedicado al pasado, presente y futuro de DIV. Muy interesante. Y una entrevista a Jose Luis Cebrían, creador de *Fenix*, un programa freeware, con el código abierto a cualquier usuario y que puede convertirse en





La inevitable sección de enlaces.

Enlaces a visitar:

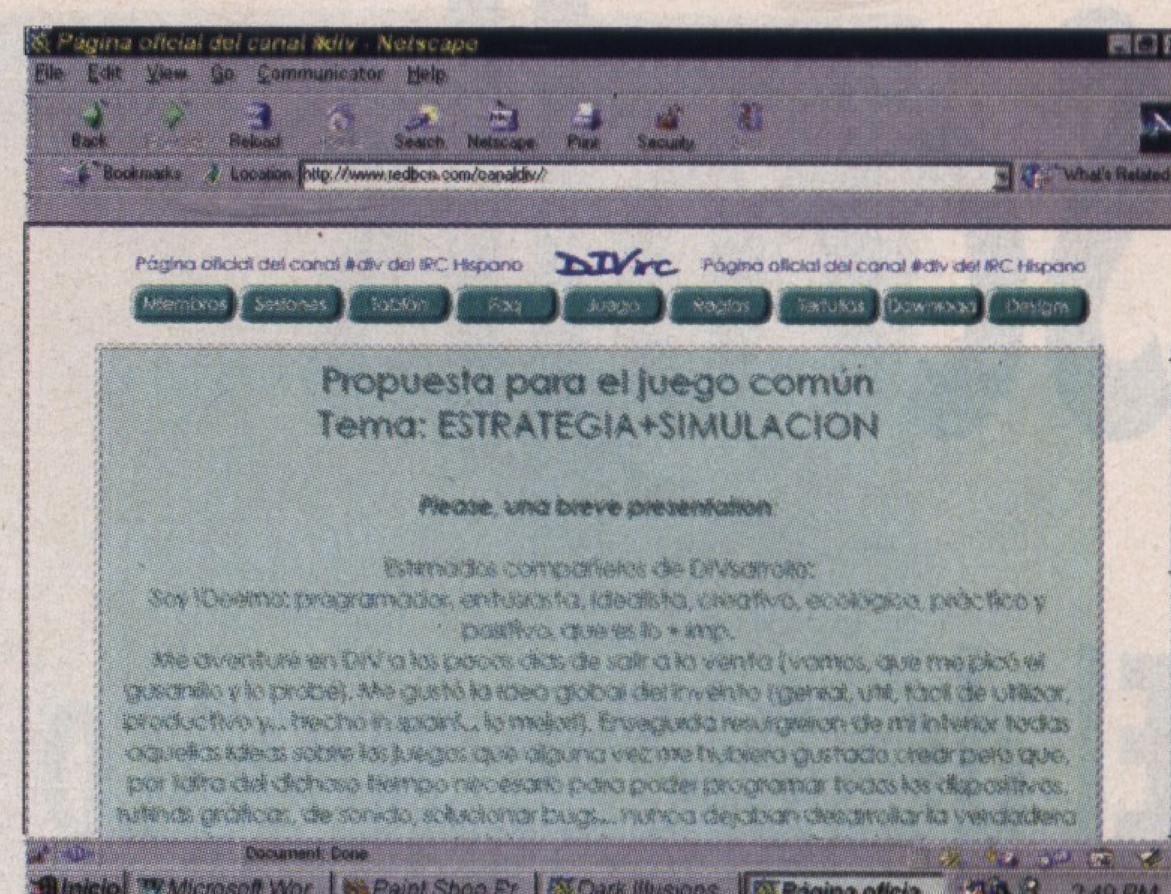
Revista electrónica "Vitalweb":
http://www.geocities.com/TimesSquare/arcade/9520

DIVnet:
http://www.pagina.de/div2

Canal de chat sobre DIV:
http://www.redbcn.com/canaldiv

Otro canal:
http://www.onelist.com/community/canaldiv

Y otro más:
http://www.fortunecity.com/roswell/philosophy/81/divchat.htm



Propuesta para la realización de un proyecto.

un futuro próximo en la alternativa a DIV 2.

Por supuesto, esta revista de distribución on-line, también tiene un rincón para recoger la opinión de los lectores y una editorial firmada por el autor de la página. Os recomendamos que le echéis un vistazo a su contenido si disponéis de Internet en casa, y si no pues bajaros al ciber-café que os quede más cerca.

Canales de chat

Hemos podido encontrar tres canales de chat dedicados a DIV, aunque suponemos que habrá más. Tenéis las tres direcciones en el cuadro adjunto.

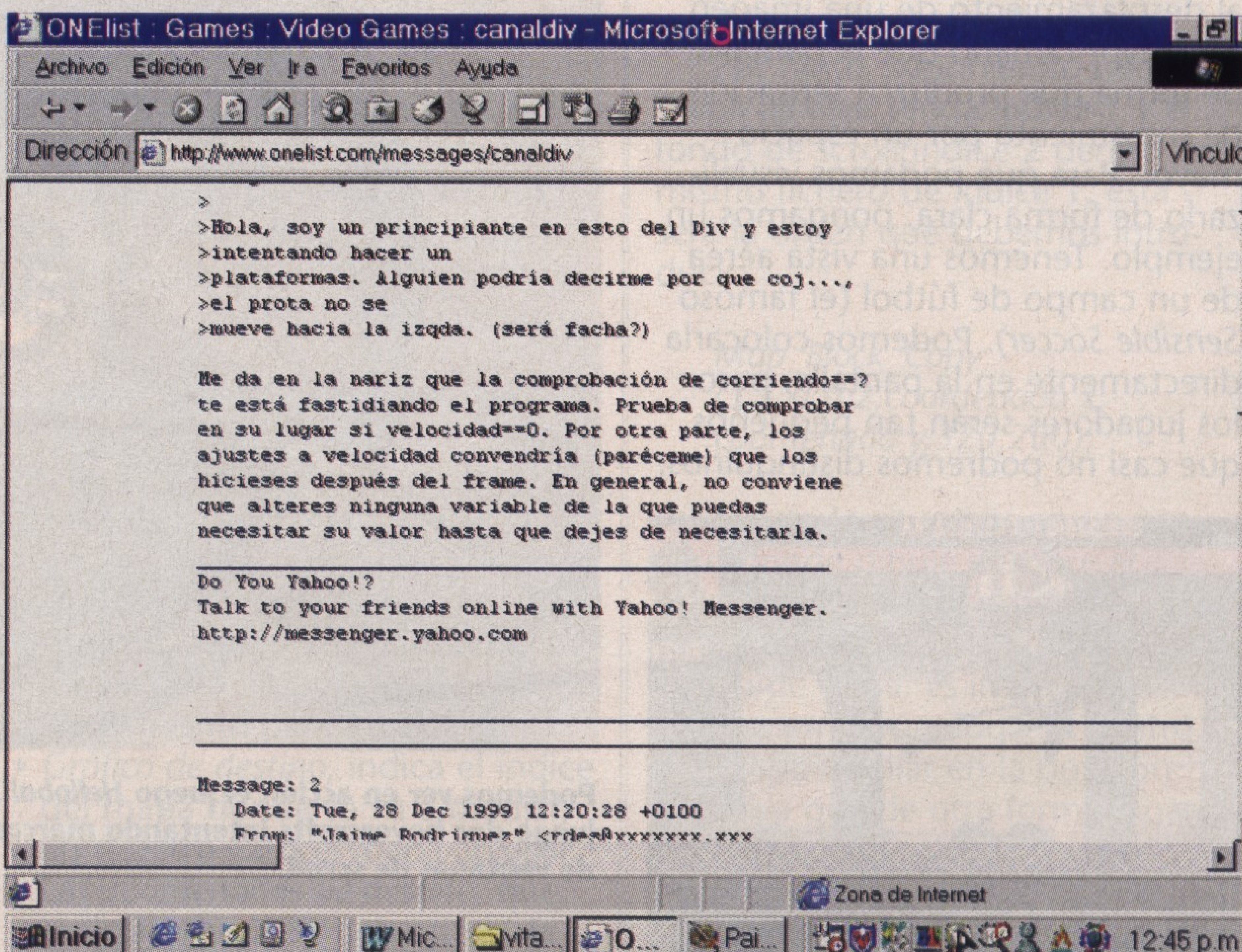
En ellos podéis encontrar, además del consabido servicio de chat, listas de correo a las que poder apuntarse para recibir información sobre la programación de juegos; un tablón de anuncios donde se indica el día y la hora para tratar un tema en concreto; propuestas de juegos en común, etc.

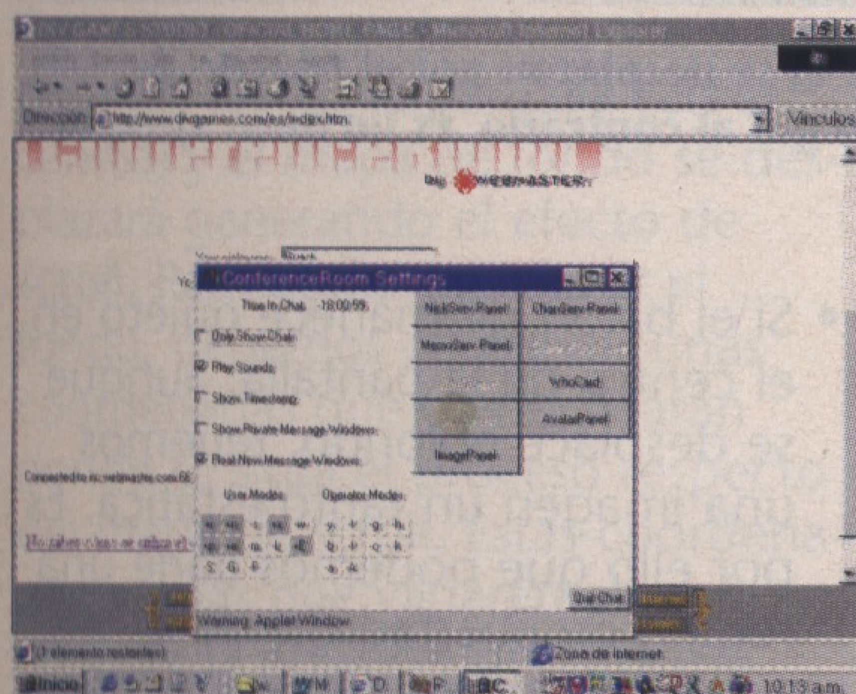
Si te interesa el mundo de la programación de juegos, pero no ves muy claro hacer un proyecto con tus escasas fuerzas, puede que conociendo a otras personas con tus mismas inquietudes te animen a trabajar en común para desarrollar un nuevo juego o para unirte a otros grupos de programación ya en pleno proceso de producción de obras de software lúdico. Si no, siempre podrás hacer nuevos amigos.

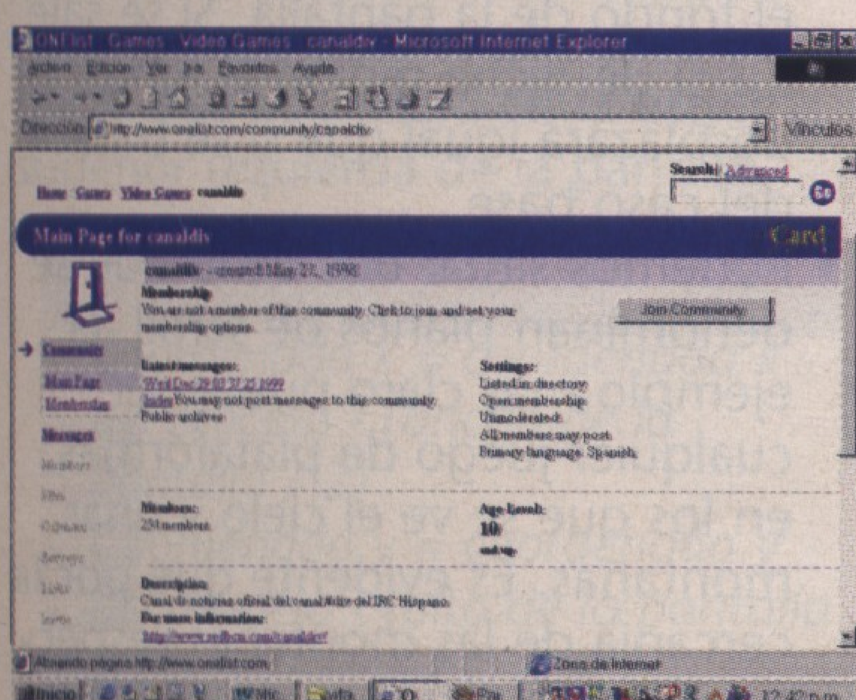
Si ya estás enfrascado en la realización de algún juego y te has quedado atascado en alguna parte del mismo, también puedes pedir ayuda. Algunos habituales de este tipo de canales de chat son auténticos monstruos de la programación y se dedican profesionalmente a crear auténticas obras de arte en forma de juegos para ordenador. No lo dudes y conéctate a cualquiera de ellos.

> Hay varios canales de chat para que los programadores noveles puedan exponer sus problemas más acuciantes

Alfredo del Barrio







Si quieres que tu página de Internet dedicada a DIV, a la programación de juegos o similares, salga reflejada en esta sección, mándanos un e-mail especificando cuál es la dirección de tu página. Esto es importante, ya que hemos recibido algún que otro correo de algún entusiasta que nos dice de todo en el mail menos la dirección que tenemos que poner en el buscador para ver su rincón de Internet.

Buscamos sobre todo dar a conocer todas aquellas direcciones que tengan algo que ver con la programación de juegos, utilicen DIV o no, a todos los aficionados a este mundillo.

La dirección a la que podéis mandar cualquier información o sugerencia es esta:

Gover@prensatecnica.com

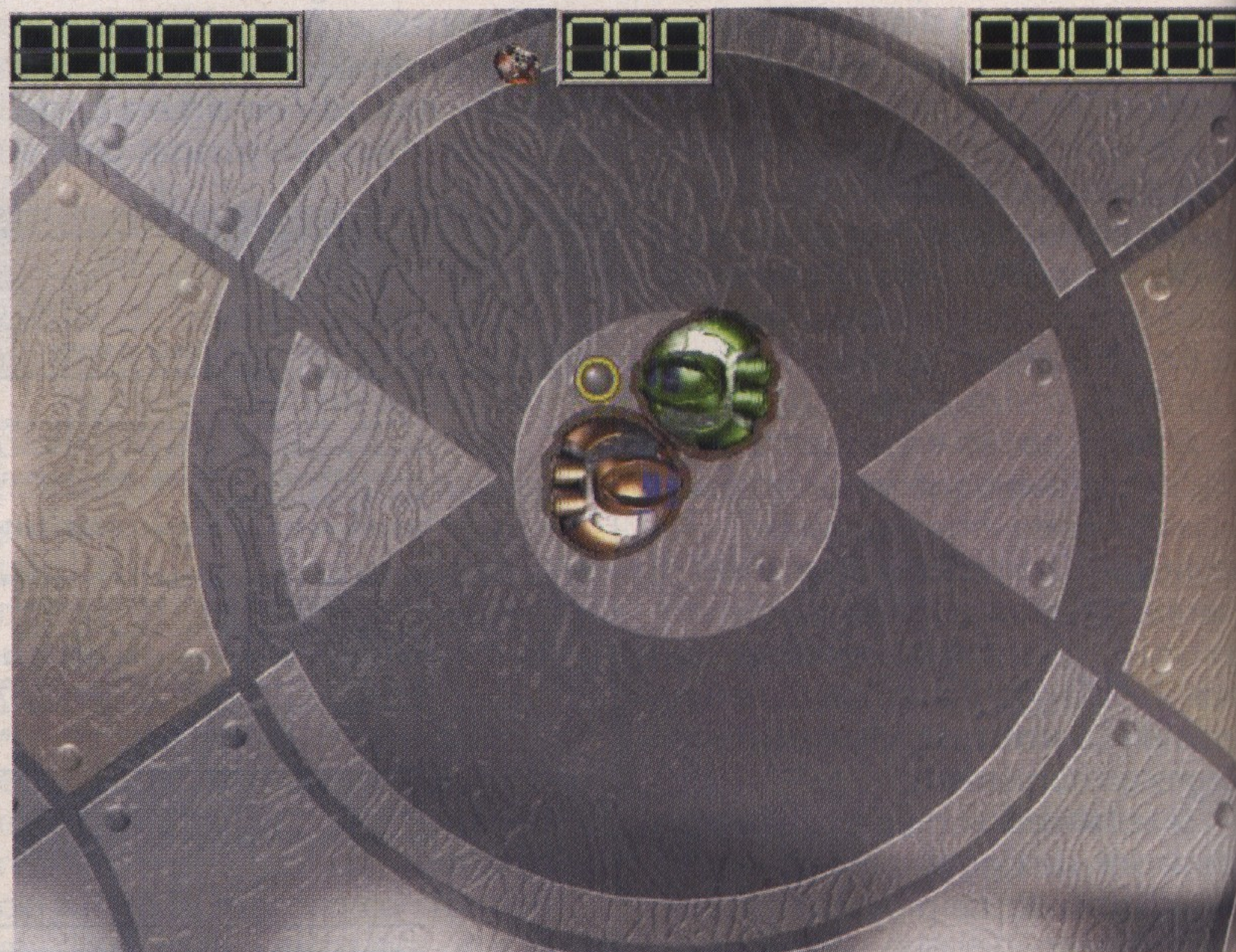
Scrolls

El desplazamiento de la pantalla

Una nave estelar se acerca hacia ti, la tienes a tiro, ya cantas victoria. Pero cuando nadie se lo esperaba, se sale de la pantalla y no puedes alcanzarla. ¿Qué ha pasado? ¿Cómo podemos alcanzar la nave? La solución pasa por los scrolls.

Muchos de ustedes se preguntarán qué significa eso de *scroll*. Con un *scroll* nos referimos al desplazamiento de una imagen de mayor tamaño que la pantalla, de forma que produzca sensación de movimiento por un espacio mayor. Para que podamos visualizarlo de forma clara, pongamos un ejemplo. Tenemos una vista aérea de un campo de fútbol (el famoso *Sensible Soccer*). Podemos colocarla directamente en la pantalla, pero los jugadores serán tan pequeños que casi no podremos distinguirlos.





Podemos ver en acción el juego *Helioball*. En él se desplazan por el terreno de juego dos hovercrafts intentando marcar un gol al contrario. Es un buen ejemplo de ventanas de scroll.

Para ello, podemos utilizar una imagen de un tamaño mucho mayor y desplazarla a la vez que el balón, que permanecerá quieto en el centro de la pantalla. De esta forma, si el balón se desplaza hacia la banda izquierda, la imagen del campo se desplazará también hacia esa zona. Con esto tenemos un efecto muy realista que nos permite aumentar el tamaño de los jugadores sin perder ningún detalle del campo.

Además de este tipo sencillo hay muchas otras variantes que nos permiten crear efectos diversos para todas las necesidades y juegos. Veamos algunas de estas mejoras:

- Si el balón permanece quieto en el centro de la pantalla, aunque se desplace el fondo, tenemos una imagen un tanto estática. Es por ello que podemos darle una región de libertad donde se puede desplazar sin que lo haga el fondo de la pantalla. Si se sale fuera de dicha región el fondo se desplazará igual que si se tratara del caso base.
- Podemos situar dos de lo que se denominan planos de scroll. El ejemplo más claro puede ser cualquier juego de plataformas en los que se ve el cielo y unas montañas. Es evidente que por la cercanía de las montañas respecto del cielo, éstas se desplazarán

más rápidamente que el cielo respecto de nosotros. Es por ello que debemos desplazar con mayor velocidad las montañas que el cielo.

- Nuestro campo de fútbol es limitado, de forma que cuando se llegue a los límites, el balón saldrá fuera y habrá un saque de banda. Sin embargo, si nos encontramos en el espacio, estos límites no existen, de forma que tenemos dos opciones:
  1. Crear un mapa de enorme tamaño, de forma que si algún curioso alguna vez osa buscar los límites tal vez lo encuentre. No es demasiado recomendable por el gran consumo de memoria y el gran tiempo que requiere hacer dicho mapa.
  2. Crear un nuevo efecto de forma que cuando alcancemos el límite nos "pegue" la pantalla de forma cíclica con el otro límite, de forma que parezca un mapa infinito. Sin lugar a dudas, este es el que debemos usar.

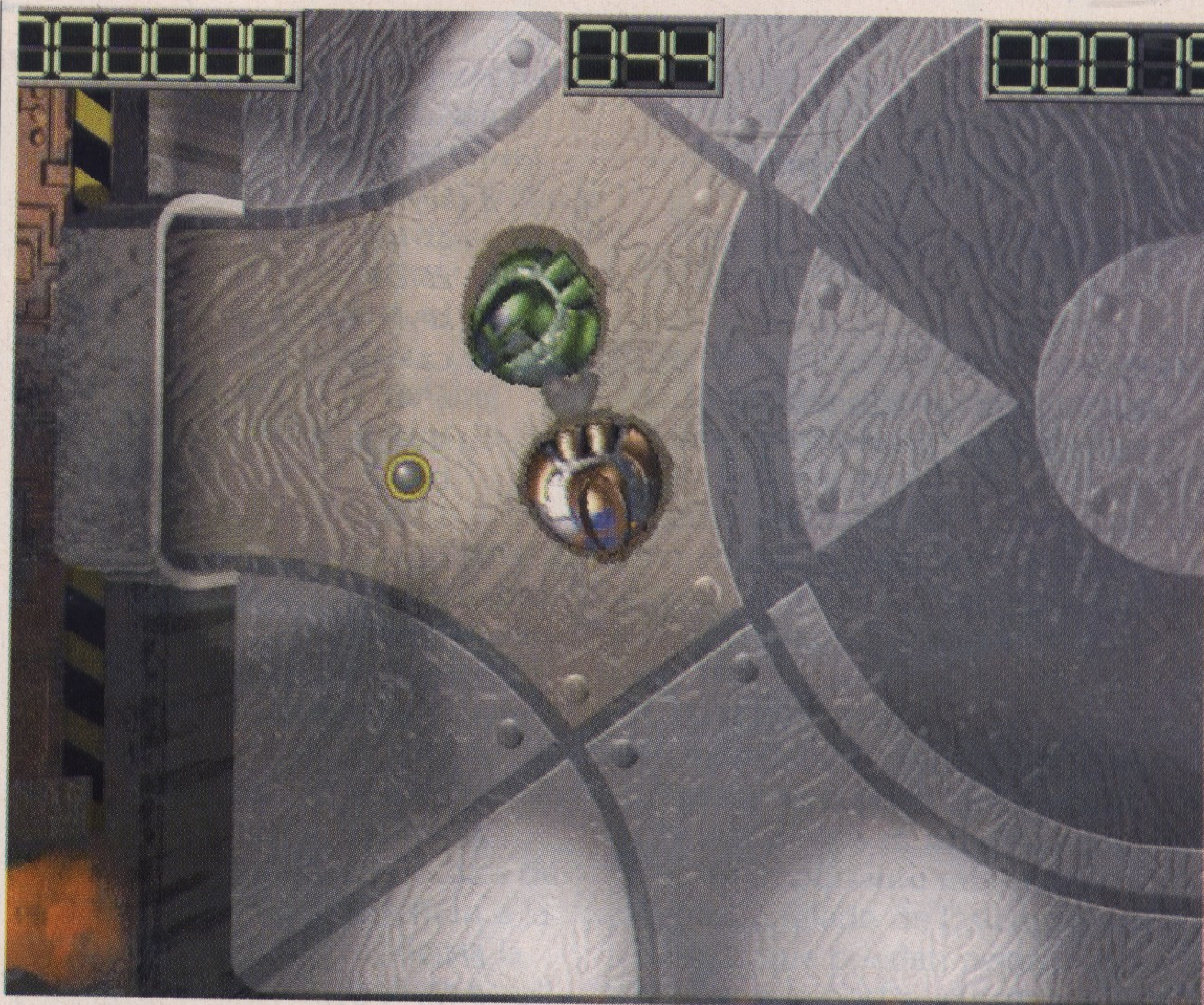
Ya que conocemos todos las posibles opciones que tenemos, pasemos a ver cómo podemos dibujarlo.

Realización de scrolls con mapas

Este método si bien no es demasiado depurado ni recomendado, ilustra perfectamente el comportamiento interno y el proceso que se sigue para generar estas imágenes.

Tenemos un mapa bien grande, de unos 800 x 600 píxeles, que por cuestiones de simplicidad, vamos a considerar finito y no cíclico, pero tan solo tenemos una pantalla de 320 x 200 píxeles. Tenemos un proceso que tiene una posición fija en la pantalla (en el centro) y el mapa de fondo se desplazará generando el efecto de scroll. Para poder desplazar el fondo, debemos conservar unas coordenadas que nos indicarán donde se sitúa el proceso respecto del mapa de scroll. Estas coordenadas son las que variaremos para conocer la posición del proceso dentro del mapa. Por tanto, podemos conocer cuál es el punto del mapa que aparecerá en la esquina superior izquierda de la pantalla aplicando estas fórmulas:

Coordenada x = coordenada x del proceso - (ancho de la pantalla / 2)
Coordenada y = coordenada y del proceso - (alto de la pantalla / 2)



El juego Helioball utiliza dos ventanas de scroll con distintas cámaras para simular un juego con dos jugadores.

Para volcarlo sobre la pantalla, en primer lugar debemos crear un mapa auxiliar del tamaño de la pantalla para dibujar sobre él el trozo del fondo de scroll que aparecerá en la misma. Para dibujar bloques de la pantalla, debemos usar la función *Map_Block_copy()*, cuya sintaxis discutiremos a continuación:

Map_Block_Copy(fichero, gráfico de destino, x destino, y destino, gráfico de origen, x, y, ancho, alto);

- El parámetro *fichero* nos indica el número del fichero donde se encuentran ambos gráficos. Debe ser el mismo para los dos.
- *Gráfico de destino*, indica el índice del mapa donde se va a dibujar, en nuestro caso el mapa auxiliar.
- Las *coordenadas de destino*, que indican donde se debe situar la coordenada superior izquierda del bloque que copiaremos. En nuestro caso serán las coordenadas (0,0), que coincidirán con la esquina superior izquierda de la pantalla.
- El *gráfico de origen* nos indica el índice del mapa del que queremos copiar el bloque, que para nosotros será el fondo de scroll.
- Las coordenadas *x* e *y* son las coordenadas donde comenzaremos a leer el bloque, y que serán las que calculamos anteriormente.
- Los parámetros *ancho* y *alto*, indican el ancho del bloque a copiar, que serán los mismos que el ancho y alto del modo de vídeo que hallamos seleccionado (320 x 200).

Por tanto, suponiendo que el mapa auxiliar tenga índice 1 y el fondo de scroll índice 2 dentro del mismo fichero de índice 1, esta será la orden que debemos introducir:

Map_Block_Copy (1,1,0,0,2,coordenada x, coordenada y,320,200);

Si esto lo situamos en un proceso donde dibuje siempre el mapa auxiliar en la posición (0,0), tendremos ya nuestro deseado efecto de scroll. Es importante colocar en primer lugar el centro del mapa auxiliar en la posición (0,0), ya que de otra forma, aparecerá el centro de dicho mapa en la esquina, con lo que mandamos al garete todo el trabajo. Recordemos que esto se hacía colocando mediante el editor gráfico el punto de control 0 en las coordenadas donde deseemos colocar el centro.

Para nuestro ejemplo hemos tomado un sencillo programa que simplemente muestra una nave que hemos seleccionado de entre la amplia librería de DIV moviéndose por un fondo de estrellas. Este fondo se ha realizado a partir de una imagen en negro a la que hemos añadido puntos blancos con el spray. Estos son los índices de las imágenes dentro del fichero de mapas: fondo (1), nave (2), mapa auxiliar (3). Veamos el código fuente de esto:

Cuadro 1. Código del programa de scrolls por volcado de mapas

```
PROGRAM scroll_con_mapas;

GLOBAL
cx,cy;

BEGIN
load_fpg("c:\div\divmania\
ejemplo4.fpg");
nave();
aux();
LOOP
    FRAME;
END
END

PROCESS nave()
BEGIN
graph = 2;
x = 160;
y = 100;
cx = 400;
cy = 300;
LOOP
// movimiento mediante
las teclas
    if (key(_left))
        cx-=2;
    END
    if (key(_right))
        cx+=2;
    END
    if (key(_up))
        cy-=2;
    END
    if (key(_down))
        cy+=2;
    END
    // controlamos que no se
    salga del fondo de scroll
    if (cx < 0)
        cx = 0;
    else IF (cx > 800-320)
            cx = 800-320;
            END
    END
    if (cy < 0)
        cy = 0;
    else IF (cy > 600-200)
            cy = 600-200;
            END
        END
    FRAME;
END
END

PROCESS aux()
BEGIN
graph = 3;
x = 0;
y = 0;
z = 50;
LOOP
    // copiamos en cada
    frame el bloque
    correspondiente
    Map_Block_Copy(0,3,0,0,1,c
    x,cy,320,200);
    FRAME;
END

END
```

Si ejecutamos el programa tendremos el efecto deseado. Es importante que en el mapa con el fondo negro, este color no tenga índice 0, ya que de esta forma, el fondo sería considerado como transparente y no volcaría el contenido adecuadamente, conservando el contenido de anteriores frames.

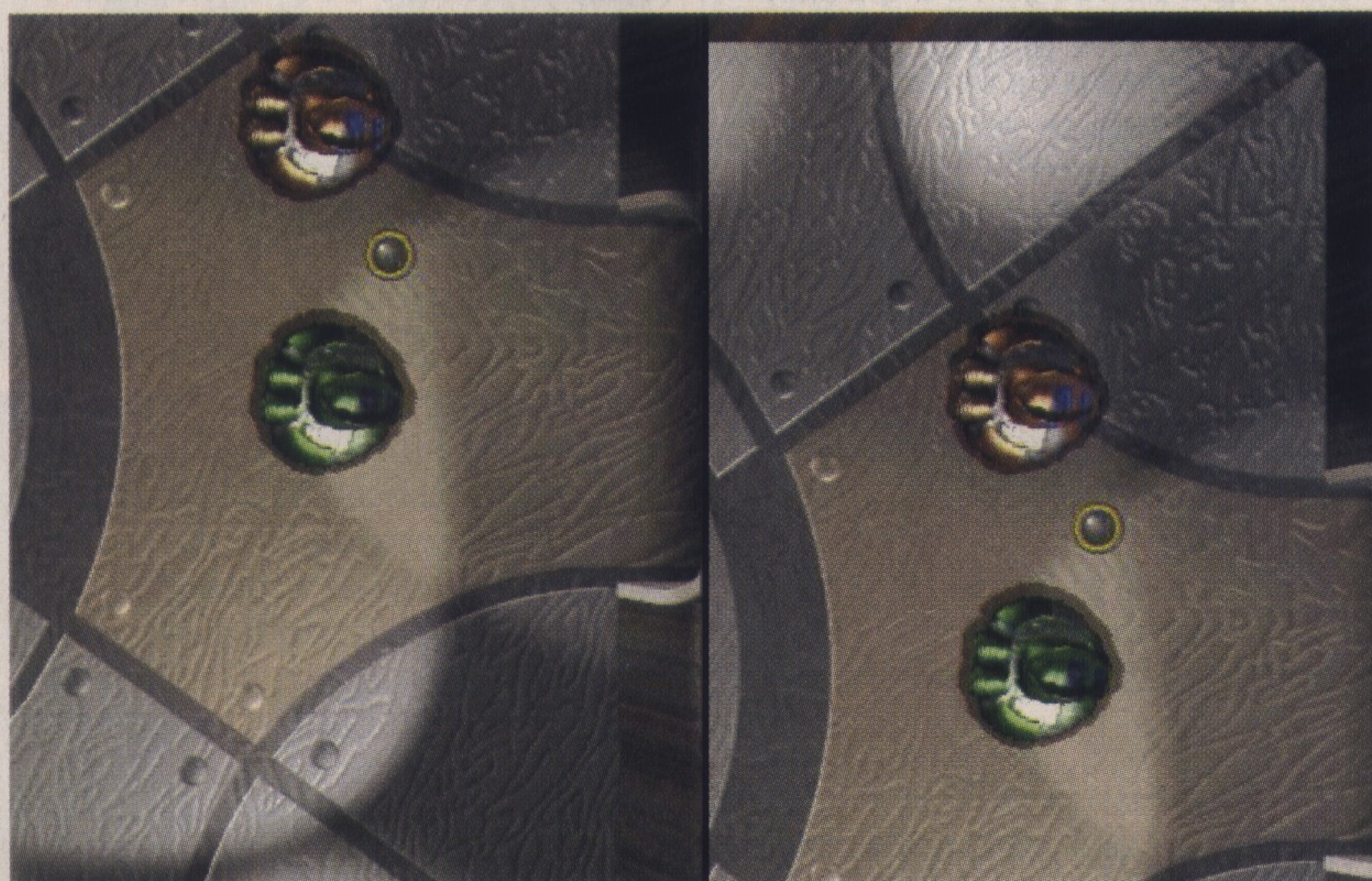
Realización de scrolls con Start_Scroll()

El proceso anterior, es completamente válido, y de hecho funciona, pero como veremos, DIV es capaz de gestionar de forma autónoma los scrolls de pantalla. Para ello utiliza la estructura global Scroll y las funciones de activación y desactivación de éstos.

En primer lugar veamos la función *Start_Scroll*, que nos permite activar lo que se denomina una región o ventana de scroll. Una región de scroll no es más que una zona de la pantalla donde se aplicará dicho efecto de scroll. DIV es capaz de gestionar hasta 10 ventanas de scroll simultáneamente. Para crear una de ellas debemos seguir la sintaxis:

Start_Scroll (<nºscroll>,<fichero>,<gráfico>,<gráfico fondo>, <region>,<indicador de bloqueo>);

- Con el primer parámetro le indicamos el número que deseamos asignar al scroll que vamos a crear, que varía entre 0 y 9.
- *Fichero* indica el índice del fichero de mapas que contiene los planos de scroll.
- *Gráfico* es el código del mapa dentro del fichero, del scroll primario que deseamos aplicar para el efecto.
- El *gráfico de fondo* es el segundo plano que podemos desear que aparezca por detrás del gráfico principal. Es importante que dicho gráfico principal tenga zonas transparentes para que se pueda ver este gráfico. Si su valor es 0, no se aplicará.
- Una región es una zona rectangular de la pantalla donde queremos aplicar un efecto. En DIV podemos crear estas zonas rectangulares con la función *define_region()*, cuya sintaxis es *define_region(indice,x,y,ancho, alto);*. Con el índice debemos especificar un número entre 1 y 31 que identificará internamente nuestra región. Esta región nos servirá para especificar la zona de la pantalla donde mostraremos nuestra ventana de scroll. Si su valor es 0, tomará toda la pantalla para representar la ventana.
- El *indicador de bloque* representa una serie de propiedades que nos permite crear planos de scroll cíclicos. Este número debe ser la suma de los valores que deseemos de la tabla 1. Si no queremos que nuestro plano de scroll tenga ninguno de estos efectos, debemos especificar este valor como 0.



El ya famoso juego de las chapas recreado en un amplio terreno utilizando de nuevo las ventanas de scroll.

Tabla 1. Valores de los identificadores de bloqueo

Valor	*Significado*
+1	*Primer plano cíclico horizontal*
+2	*Primer plano cíclico vertical*
+4	*Segundo plano cíclico horizontal*
+8	*Segundo plano cíclico vertical*

Además de esta llamada, debemos definir otros valores para determinar de forma definitiva nuestra ventana de scroll. Estos valores se encuentran en la estructura global *scroll*. Esta estructura nos permite determinar estos valores en cualquiera de las 10 ventanas indicándole el índice de dicha ventana de scroll de la siguiente manera:

Scroll[indice]

Podemos suprimir el índice y los corchetes para referirnos al scroll número 0. Veamos algunos valores de la estructura *scroll*:

- *x0,y0*: indican las coordenadas de la esquina superior izquierda de la ventana de scroll. Variando estos valores podremos desplazar la imagen de fondo.
- *x1,y1*: idéntico al anterior pero para el segundo plano.
- *Camera*: si inicializamos este valor con el identificador de un proceso, la ventana de scroll seguirá permanentemente el movimiento de éste.
- *Ratio*: indica la proporción del desplazamiento del primer plano respecto del segundo en porcentaje. Por ejemplo un valor de 200 indicará que el primer plano se desplazará el doble de rápido que el segundo.
- *Region1*: si se inicializa con un índice entre 1 y 31 (de una región previamente creada con la función *define_region();)*, nos indicará la zona de la ventana o región de libertad donde se desplazará el proceso asociado y no el fondo.

Las dos últimas variables *(ratio y region1)* sólo tienen efecto si hemos determinado la ventana como scroll automático, es decir, que persiga a un proceso, o lo que es lo mismo, iniciar la variable *camera* con el identificador de dicho proceso.

Pero esto no es todo lo que precisamos para realizar efectos de scroll. Para que DIV sepa qué procesos deben mostrarse en las ventanas de scroll, debemos especificar el valor de la variable local *ctype* de todos los procesos de la siguiente forma:

Ctype = c_scroll;

Así, las coordenadas locales de estos procesos no se referirán a la posición en la pantalla, sino a la posición relativa al plano de scroll en el cual se desplaza. Por tanto, la forma de mover el fondo de scroll en una ventana donde tenemos un proceso cámara, es variando las variables *x* e *y* de dicho proceso.

Nada mejor que ver un ejemplo que nos ilustre cómo realizar el mismo scroll que hicimos anteriormente mediante mapas. **Cuadro 2.**

Al comienzo fijamos los valores que definen nuestra ventana rellenando los campos adecuados de la estructura *scroll*. Estos serán las coordenadas de la esquina superior izquierda del mapa de estrellas, así como el identificador de la nave, que obtendremos de la creación de dicho proceso y utilizaremos para dar el valor adecuado al campo *camera*.

Podemos comprobar cómo este programa realiza lo mismo que el que desarrollamos con anterioridad. Pero ahora queremos añadirle una región de libertad donde se desplazará libremente el proceso. Para ello, debemos utilizar la región que la define dentro de la estructura *scroll*, que es *region1*. Esta región irá desde las coordenadas (100,50) a (220,150). Es tan sencillo como añadir este código:

define_region(2,100,50,120,100);
scroll.region1 = 2;

Si ahora comprobamos el resultado de esta operación, veremos como simplemente añadiendo un par de líneas hemos conseguido un efecto bastante interesante. Esto sin embargo sería mucho más complicado con el primer método visto con anterioridad.

Conclusiones

Hemos tratado en este número una interesante técnica que nos permitirá crear juegos mucho más dinámicos. Es importante saber que sólo hemos hablado de los conceptos básicos que pueden permitirnos realizar a la postre, efectos de scroll mucho más complejos. Para poder profundizar mucho más en este tema, es muy recomendable consultar el código fuente de juegos que utilicen esta técnica para ver cómo la aplican. Un buen ejemplo puede ser el programa *Helioball* o *World Chapas Championship*, que acompañan al CD de DIV. Para cualquier consulta, opinión o duda, puede escribir un e-mail a trinidad@arrakis.es

Pablo Trinidad



El editor gráfico en acción.

Cuadro 2. Código del programa de scroll

```
PROGRAM naves;

GLOBAL
id_nave;

BEGIN

load_fpg("c:\div\divmania\ej
emplo4.fpg");
scroll.x0 = 0;
scroll.y0 = 0;
id_nave = nave();
scroll.camera = id_nave;
start_scroll(0,0,1,0,0,3);
LOOP
   FRAME;
END

END

PROCESS nave()
BEGIN
ctype = c_scroll;
graph = 2;
x = 100;
y = 100;
LOOP
   if (key(_left))
      x-=2;
      END
   if (key(_right))
      x+=2;
      END
   if (key(_up))
      y-=2;
      END
   if (key(_down))
      y+=2;
      END
   FRAME;
 END
 END
```

Archivos de mapas

Respuesta a todas tus preguntas

Los mapas son el pan de cada día en el mundo de DIV. Sabemos manejarlos, pero, ¿cómo se almacenan dichos mapas? ¿Cuál es el formato? ¿Sabiendo su formato, puedo hacer un programa que manipule estos archivos?

Hasta ahora, pocos se habían dado cuenta de la cantidad de ventajas que podemos obtener en el desarrollo de juegos en DIV mediante el uso de herramientas externas. Aplicaciones como un generador de plataformas, generación de código por módulos y quizás el más destacado, el DIVC (un nuevo compilador de DIV alternativo que permite ejecutar los programas en el entorno Windows 95 y Linux), son claros ejemplos de las posibilidades que se abren ante nuestros ojos. Es mi intención por ello, animar a todos los lectores para que hagan sus propias utilidades y herramientas que potencien las capacidades de DIV. Por ello, en este número, vamos a realizar un lector de archivos MAP, que por razones de comprensión y complejidad hemos optado por reducirlo al máximo, pero que con poco trabajo, podemos convertir en, por ejemplo, un editor de archivos MAP. Utilizaremos para este lector de mapas el compilador Visual C++ para el entorno Windows 95, aunque el código que nos interesa, viene a ser compatible con cualquier otro compilador de C++ o sistema operativo haciendo pocos cambios. Sin más preámbulos, pasemos a la acción.

Se anima a los lectores a realizar sus propias utilidades para el programa DIV

Conociendo el formato

El formato MAP, es bastante sencillo en su estructura, sobre todo una vez conocido el formato PCX. Los datos de la imagen no tienen compresión alguna, con lo que cada punto en la pantalla, se corresponderá con un byte en el fichero. Es por ello que nuestra principal preocupación será la correcta localización del comienzo de los datos de la imagen y una buena organización de la paleta. Podemos ver en la tabla 1, la descripción de este formato gráfico. Como podemos observar, disponemos de una serie de datos que nos permiten comprobar si el formato con el que vamos a tratar es el correcto: el identificador, el código y la versión. Es conveniente hacer dicha comprobación antes de manipular cualquier fichero MAP, por los posibles errores (que no son pocos) que podría causar tratar cualquier otro fichero como de tipo MAP.

Dentro de la cabecera nos encontramos con otro dato importante, el tamaño del mapa, es decir, los puntos de ancho y alto que tiene el mapa. Esto nos servirá más adelante para saber el número de puntos del gráfico que debemos leer. Para terminar con la cabecera, disponemos del código del gráfico que lo identifica, y de una descripción en forma de texto ASCII que son los que aparecerán en un archivo FPG.

Una vez analizadas las principales características de nuestra imagen, pasamos a ver los valores de la paleta. Podemos comprobar el formato de la paleta en la Tabla 2. Este formato es el mismo que se utiliza en los formatos de paleta PAL. Hemos de recordar que los valores RGB contenidos en este formato oscilan entre 0 y 63, con lo que para manipularlos correctamente en cualquier entorno distinto a DIV, debemos multiplicar por 4 el valor obtenido.

Los puntos de control nos permiten tener localizado el centro de la imagen, así como otros puntos importantes del mapa. Tras esto podemos proceder al análisis de la imagen. Por cada punto obtenemos un valor de 0 a 255 que nos indica el índice del color dentro de la paleta de colores.



DIV developer

NÚMERO 6

Sumario



Curso de programación y concurso

Continuamos con nuestros cursos de programación habituales siguiendo los capítulos ya iniciados. Como siempre, esperamos que os sean de utilidad. En cuanto a nuestro concurso, por supuesto sigue en pie con los mismos premios. Ya sabéis: 25.000 pesetas para el ganador y 20.000 para los otros dos juegos elegidos. Esperamos con impaciencia recibir vuestros juegos. No olvideis adjuntar todos vuestros datos.

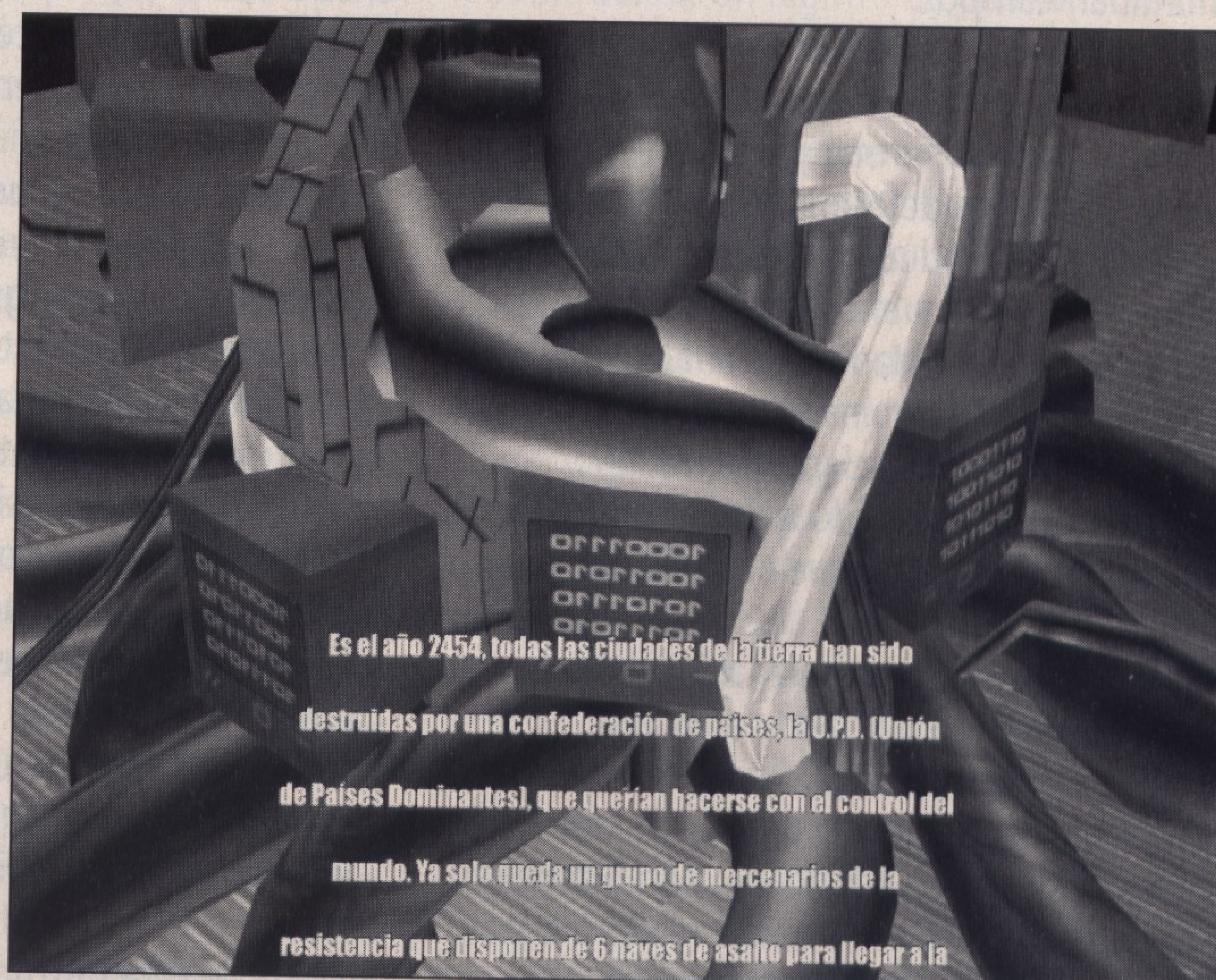
Aquí tenéis los ganadores de este número. Esperamos que os gusten. En este suplemento encontraréis un breve comentario del juego y un extracto de su código fuente que, no os olvidéis podéis encontrar entero en el CD-Rom si abrís los ficheros pertinentes. Seguro que la visión y el estudio de estos códigos os ayudará a mejorar algunos aspectos de vuestros propios trabajos de programación.

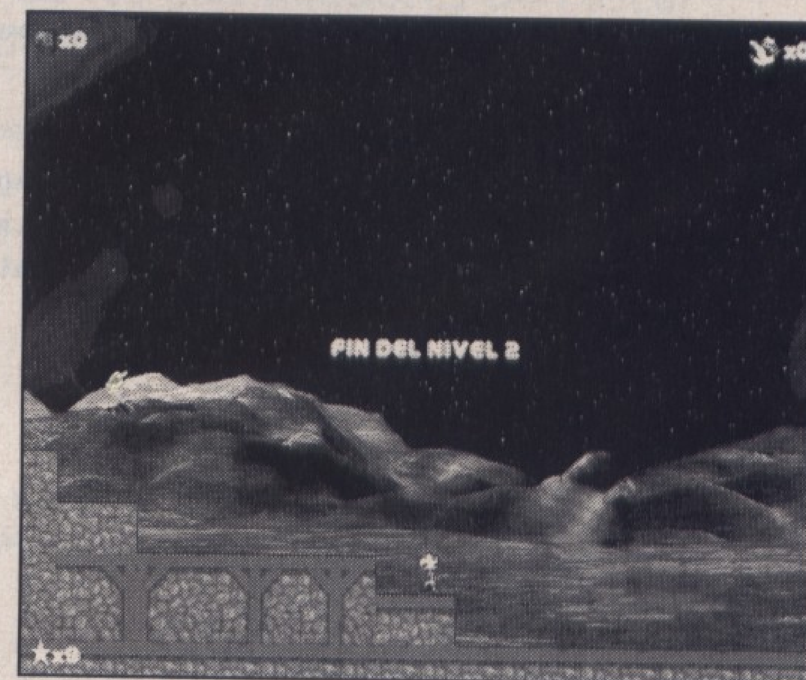
Nos gustaría aclarar un punto sobre nuestro concurso. Los ganadores se eligen entre los juegos recibidos desde el cierre del número anterior hasta el cierre del siguiente. Por este motivo, en ocasiones nos vemos obligados a dejar en la cuneta juegos que realmente merecen la pena y, en otras, debemos elegir entre otros que, tal vez, no sean tan buenos como los de las ediciones anteriores. Este es uno de los motivos que nos han llevado a publicar en el Cd todos los juegos que hemos recibido para participar en el concurso. La lista de juegos, con su correspondiente autor, la encontraréis en las páginas de contenido del Cd.

Vuestras sugerencias

Queremos haceros saber que todas las ideas, críticas o sugerencias para mejorar esta revista serán bienvenidas por parte de esta redacción. Si tenéis inquietud por algún tema que no tocamos demasiado, si consideráis que alguna sección de esta revista debe desaparecer o debe ser cambiada por otra, incluso si os animáis a proponernos artículos realizados por vosotros/as, que creéis pueden ser aprovechados por otros usuarios/as, no dudéis en poneros en contacto con nosotros. Todas las sugerencias serán atendidas y tenidas en cuenta. Os necesitamos para mejorar.
Para mandarnos los juegos o cualquier sugerencia sobre la publicación, podéis hacerlo por e-mail, correo normal o incluso entregarlos en mano en la dirección de la redacción:

Nuestra dirección:
gover@prensatecnica.com
Divmanía
C/ Alfonso Gómez, 42
Nave 1-1-2
28037, Madrid, España





Destacamos

En nuestro CD incluimos los juegos que han resultado ganadores en este número y, cómo no, sus respectivos códigos para que veáis todo el proceso de creación. Además, os ofrecemos un especial juegos en el que encontraréis más de 50 programas realizados con Div Games Studio, que se han presentado a nuestro concurso de programación. Esperamos que os DIVirtáis jugando con ellos.

Registros y ficheros

La información contenida en un DNI no es posible almacenarla en una estructura de tipo *array* de forma natural, pues ésta presenta datos que no son del mismo tipo y no sería conveniente acceder a cada uno de los datos mediante la posición que ocupa dentro de la estructura.
Los lenguajes de programación disponen del tipo de datos registro (record) que sí permite agrupar datos de tipos totalmente diferentes en una única estructura.

Los registros son estructuras de datos utilizadas para agrupar datos elementales de diferente tipo y que se encuentran relacionados. Normalmente, los registros se utilizan para dotar de una estructura a la información que se almacenará en ficheros con tipo.

REGISTROS

Un registro no es más que una estructura de datos que se compone de datos más sencillos a los que agrupa. Cada uno de los datos que forman parte de un registro se denominan campos.
En la figura 1 se muestra un ejemplo de definición de un registro para almacenar los datos que podrían formar parte de la descripción de un determinado producto. El tipo de datos "producto" contiene cuatro campos que reflejan el código, la descripción, la cantidad almacenada y el precio para un determinado producto.
Posteriormente a la declaración del tipo producto se ha procedido a declarar dos variables de dicho tipo que contendrán éstos cuatro datos.
Cuando se referencia una variable de tipo registro se está referenciando a toda la información que contiene. Para poder acceder por separado a cada uno de sus campos se deberá especificar el nombre del campo precedido de la variable registro a la que pertenece separados entre ellos por un punto. El dato resultante será del mismo tipo que el del campo, lo que implica que cualquier referencia a un campo puede aparecer donde pueda hacerlo una variable de dicho tipo. Así, en la figura 1 al campo *cant_almacenada* se le puede asignar el valor entero 5, o bien, se puede comprobar si el campo *cod_producto* contiene un determinado código en la condición asociada a una instrucción *if*.

```
type
 producto = record
                cod_producto : string(20) ;
                dscr_producto : string(80) ;
                cant_almacenada : integer ;
                precio : integer ;
              end ;
var
 producto_1, producto_2 : producto ;

...

begin
...
producto_1.const_almacenada := 5 ;
if producto_2.cod_producto = "03456AF"
  then
...
end.
```

Figura 1.

Es posible no tener que referenciar la variable de tipo registro a la que pertenece un determinado campo gracias a la existencia de la instrucción *with.* Esta instrucción seguida de un nombre de variable de tipo registro permite cualificar dentro de su ámbito un conjunto de campos sin tener que especificar el nombre de la variable registro a la que pertenecen. Es útil, cuando en líneas adyacentes se van a realizar referencias a los campos de un mismo registro, o bien, cuando a lo largo de un trozo de código se va a referenciar de forma continuada a uno o varios campos. En la figura 2 se muestra la utilización de esta instrucción; las reglas de ámbito de esta instrucción son las mismas que para otras instrucciones.
Puesto que un registro es un tipo de datos es posible que un campo de un registro sea a su vez un registro; a este tipo de registros se les denomina registros anidados. El comportamiento sigue siendo el habitual pero, para acceder a los campos de los registros definidos más internamente, se deberá especificar el registro más externo al que pertenece, junto con el registro que es a su vez un campo de dicho registro.

REGISTROS CON VARIANTES

Los registros anteriormente descritos presentan siempre los mismos campos, aunque éstos pueden ser de diversa naturaleza, sin embargo, es posible construir registros que presenten diferentes tipos de datos dependiendo de una parte fija; se trata de los registros con variantes.

Se debe especificar el nombre del campo precedido de la variable registro

Supóngase la siguiente situación: normalmente los productos representados por el registro de la figura 1 se fabrican en la misma empresa, sin embargo, hay una mínima parte de ellos que son fabricados por empresas externas. Para representar éste hecho, es posible incluir dos campos más en el registro de tal forma que se pueda registrar el nombre y la dirección de la empresa que los fabrica. Ésta solución sería ineficaz pues la mayoría de éstos campos quedan vacíos debido a que casi todos los productos son fabricados por la propia empresa, por lo que se derrocharía espacio al almacenarlos. Además, se tendría que realizar un tratamiento especial para aquellos registros que tuviesen la correspondiente información almacenada.
Para evitar situaciones como la anteriormente descrita es posible definir registros con variantes, de tal forma que en base al contenido de un campo del registro pueda decidirse si los campos que son variantes u opcionales deben ser tratados o no.
En la figura 3 se muestra la solución dada a través de un registro variante: a través del campo de tipo booleano *otra_empresa* es posible decidir si los datos del fabricante del

```
var
 producto_1, producto_2 : producto ;
...

begin
...
with producto_1
     do
      begin
        cod_producto := "32433SD" ;
       dscr_producto := "Martillo" ;
       cant_almacenda := 32 ;
       precio := 350 ;
      end ;
...
end.
```

FIGURA 2.

```
type
 producto = record
                  cod_producto : string(20) ;
                  dscr_producto : string(80) ;
                  cant_almacenada : integer ;
                  precio : integer ;
                  case otra_empresa : boolean of
                   true :(denom_empresa : string(60) ;
                          dir_empresa : string(80) ) ;
                 end ;
var
 producto_1 : producto ;
begin
 with producto_1
     do
      begin
       cod_producto := "4354YY" ;
      dscr_producto := "caja de clavos" ;
       cant_almacenada := 50 ;
       precio := 50 ;
       otra_empresa := true ;
       denom_empresa := "Tala S.L." ;
       dir_empresa := "C\ Suspiro Verde n° 4" ;
      end ;
 ...
 if otra_empresa
    then
      write ('La empresa fabricante es ', producto_1.denom_empresa')
    else
      write ('Este producto es fabricado por nuestra empresa') ;
end.
```

FIGURA 3.

producto estarán disponibles o no. Si el contenido de dicho campo es ***true*** será posible acceder al contenido de los campos *denom_empresa* y *dir_empresa*, por el contrario, si este campo tuviese el valor ***false***, entonces no se podría acceder a dichos campos.

No es muy aconsejable utilizar registros si se piensa usar un fichero para almacenarlos

El campo que sirve para discriminar la existencia o no de otros campos (en el ejemplo *otra_empresa*), puede ser de cualquier tipo subrango permitiéndose hasta 256 valores diferentes. De forma general, se podrán identificar tantos grupos de campos como valores pueda tener el campo discriminador. De todas formas, no es muy aconsejable utilizar este tipo de registros si se piensa utilizar un fichero para almacenarlos, pues es preferible mantener la información separada. Si se analiza el anterior ejemplo, el que existan una serie de campos variantes en el registro es debido a que se están almacenando datos de diferente naturaleza: por un lado, la información referente a un producto y, por otro, la empresa que lo fabrica. Los problemas que plantea el mantener la información almacenada mediante un registro variante es la siguiente:

- Existe información duplicada, pues cuando dos productos son fabricados por la misma empresa aparecerá repetida su dirección.
- La posibilidad de que existan inconsistencias en la información es mayor, pues ¿qué ocurriría si para dos productos fabricados por la misma empresa ésta tuviese direcciones diferentes?.
- La información relativa a una empresa depende de que exista un producto fabricado por dicha empresa, aún cuando fuera interesante mantener dichos datos.

Todos estos inconvenientes pueden ser eliminados utilizando dos registros que almacenasen los datos separados para la empresa y el producto, teniendo el producto un campo extra que contuviese el código de la empresa que lo fabrica. Si se deseara buscar la dirección de la empresa que fabrica un determinado producto, bastaría con mirar el registro de empresas que tuviese por código de empresa el correspondiente al del producto.

Dentro de los registros con variantes existen los denominados de enlace libre, cuya diferencia con el resto de registros variantes es que no disponen de un campo para discriminar el tipo de datos que almacenan, permitiendo representar un mismo dato de diferentes formas.

Supóngase que se desea enviar por un puerto de E/S una serie de números enteros. Todos los puertos de E/S tratan los datos en unidades de byte lo que implica que si se desea enviar la información correspondiente a un entero habría que realizar una serie de operaciones para trocear el entero en bytes. Aunque esto no ofrece ninguna dificultad aplicando las operaciones aritméticas correspondientes, es posible utilizar un registro de tipo variable con enlace libre que va a permitir que el dato entero disponga de ambas representaciones.

En la figura 4 puede observarse cómo el registro *puerto* es un registro con variantes pero sin campo selector por lo que no puede contener diferentes estructuras dependiendo del valor de un campo; este registro sólo puede contener en un momento dado un número entero, o lo que es equivalente, dos bytes. Cuando se desea almacenar un número entero se utiliza el campo *valor_entero* asociado a la cláusula *true*; cuando se desea acceder a la información en forma de byte se utilizan los campos *byte_alto* y *byte_bajo* asociados a la cláusula *false*.

Una razón de peso a la hora de utilizar estos registros es la comodidad para el programador y la transportabilidad, pues la mayoría de los compiladores de Pascal poseen éste tipo de registros.

```
type
     dual= record
           case boolean of
              true : (valor_entero : integer) ;
                   false : (byte_alto : byte ;
                                 byte_bajo : byte ) ;
          end ;
var
 puerto : dual ;
 valor : integer ;
...
begin
...
 valor := 124 ;
 puerto.valor_entero := valor ; {representación como entero}
 port[$A0] := puerto.byte_alto ; {representación como byte}
 port[$A0] := puerto.byte_bajo ; {representación como byte}
...
end.
```

FIGURA 4.

Paso y recepción de valores en función bajo C

En este capítulo van a continuarse las explicaciones que se empezaron en el anterior número del curso sobre el uso de las funciones, mostrando en esta ocasión la forma en que se pueden enviar y recibir valores hacia y desde una función.

En el anterior capítulo, como muchos recordarán, se detalló a fondo la forma mediante la que poder definir nuestras propias funciones para utilizarlas en nuestros programas. Ahora que el lector ya tiene la base en este asunto, se va a ampliar el tema del uso de funciones, detallando la forma en que se pueden definir funciones que reciban tantos valores como queramos, para poder de esta forma crear subrutinas, mucho más útiles y versátiles.

USO DE ARGUMENTOS PARA PASAR DATOS A UNA FUNCIÓN

Continuando el curso donde lo dejamos el pasado número, tenemos que las funciones que se enseñaron a usar eran poco flexibles, ya que las llamábamos y ellas hacían lo que tenían que hacer y devolvían un valor o no. Pero con este sistema de funcionamiento, nos encontramos con que no podemos influir en la función.

Los argumentos pueden indicarse tanto como variable como con valores inmediatos

Llegados aquí nos encontramos también con que sería interesante tener un poco más de control sobre lo que hacen las funciones. El mecanismo utilizado para comunicar información a una función es el llamado *argumento* (a menudo se usa el término parámetro, que quizás le es más familiar al lector). El lector en realidad ya ha usado argumentos en funciones explicadas en anteriores capítulos, más concretamente en las *printf( )* y *scanf( )*. Las cadenas de formato y los valores utilizados dentro de los paréntesis constituyen los argumentos de estas funciones.
A continuación, se da un ejemplo de programa en el que se pasa un argumento simple a una función:

```
/* grafbar.c */
/* dibuja gráficos de barras, comprobando los
argumentos de una función */
void bar(int);

main( )
{
  printf("Juan\t");
  bar(27);
  printf("Cristina\t");
  bar(41);
  printf("Carlos\t");
  bar(34);
  printf("Ana\t");
  bar(22);
  printf("Enrique\t");
  bar(15);
}
/* bar() */
/* función que dibuja barras horizontales de log
"puntuacion" */
void bar( int puntuacion)
 {
  int j;
  for(j=1; j<puntuacion; j++)
    printf("\xCD");
  printf("\n");
 }
```

El programa mostrado genera un gráfico de barras con nombres y barras que representan las puntuaciones obtenidas en una hipotética prueba de ortografía.
En este programa, el propósito de la función *bar* es el de dibujar una línea horizontal en la pantalla, construida con caracteres gráficos de doble línea *('\xCD')*. Para cada persona (Juan, Cristina...) el programa principal escribe un nombre y luego llama a la función, utilizando como argumento la puntuación obtenida en la hipotética prueba de ortografía.

ESTRUCTURA DE LAS LLAMADAS A UNA FUNCIÓN CON ARGUMENTOS

Hay que destacar ciertos aspectos sobre este programa. Primero el número que queremos pasar a *bar( )* desde el programa principal está incluido en los paréntesis que acompañan a la función *bar* en la llamada a función:

```
bar(27);
```

Prodríamos haber usado un nombre de variable en lugar de la constante 27; veremos en breve un ejemplo de este tema.
El declarador, que se encuentra al principio de la definición de la función *bar( )*, consta del nombre de la función, del nombre de la variable (puntuación) y de su tipo, estos dos últimos incluidos entre paréntesis.

```
void bar(int puntuacion);
```

Se puede ver que el valor puntuación se transfiere como por arte de magia a la función.
El valor colocado entre paréntesis en la llamada a la función se asigna automáticamente a la variable puntuación de *bar( )*.
El prototipo refleja el tipo de dato del argumento:

```
void bar(int);   /* prototipo */
```

Observe que el prototipo se diferencia del declarador en que aquél no utiliza nombre

res

ones que se
re el uso de
na en que se
a función.

rgumento la
hipotética prueba

AMADAS A UNA
NTOS

ectos sobre este
o que queremos
ma principal está
acompañan a la
nción:

nombre de
nte 27; veremos
tema.
ra al principio
bar( ), consta
nombre de la
tipo, estos dos
esis.

ación se
gia a la

sis en la
comáticamente

to del

rencia del
a nombre

de variable para el argumento, sólo el tipo de dato (se puede utilizar un nombre, como veremos pronto, pero es opcional). Como la función *bar( )* no devuelve nada, su tipo es *void*.

A los valores que se pasan a una función se les llama argumentos

La variable *puntuacion* en la función *bar( )* no está declarada en la misma forma que las variables normales. No obstante, se la declara; su inclusión en el declarador de la función sirve no sólo para especificar que es un argumento de una función, sino para declararla. Puede ser utilizada como cualquier otra variable de la función; la única diferencia es que el programa le asigna un valor inicial cuando llama a la función.

PASO DE VARIABLES COMO ARGUMENTO

En el ejemplo anterior hemos pasado a la función *bar( )* constantes (tales como el número 27) como argumentos. Podemos también pasar una *variable* desde el programa a la función, como demuestra esta variación del programa *grafbar.c*:

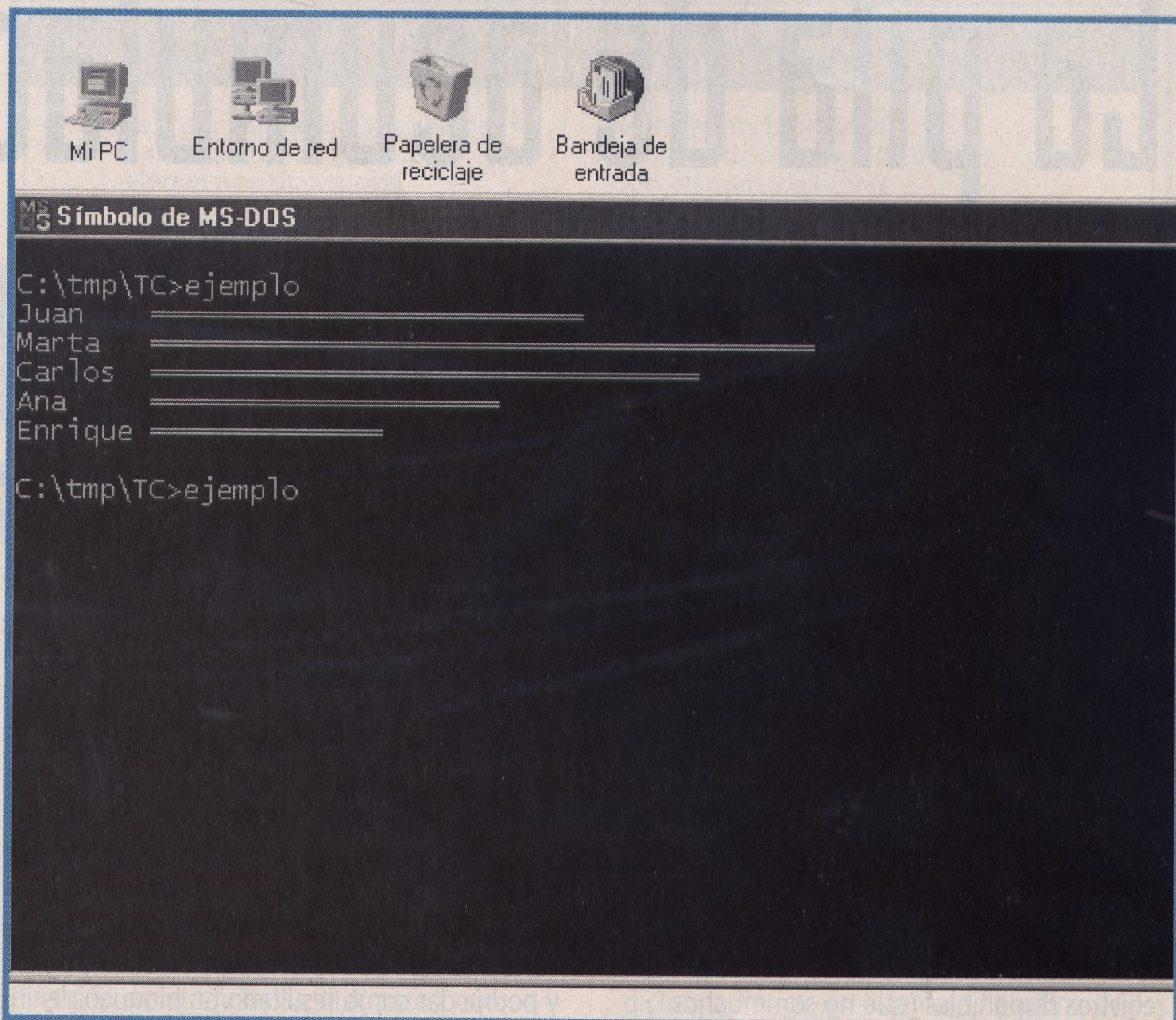
```
/* grafbar2.c */
/* dibuja gráficos de barras, comprobando los
argumentos de una función */
void bar(int inpunt);

main( )
{
 int inpunt;

 while(1)
  {
   printf("Puntuación=");
   printf("%d",&inpunt);
   bar(inpunt);
  }
}

/* bar( ) */
/* función que dibuja barras horizontales */
void bar(int puntuación)
{
 int j;

 for(j=1;j<puntuacion; j++)
   printf("\xCD");
 printf("\n");
}
```



EJEMPLO DE EJECUCION DEL PROGRAMA QUE DIBUJA UNA ESTADISTICA GRAFICA DE DATOS EN PANTALLA.

En este programa, la función *bar( )* es la misma que antes. El prototipo es también el mismo. Sin embargo, el programa principal ha sido modificado para aceptar las puntuaciones introducidas desde el teclado. Como las puntuaciones que pasamos a *bar( )* no son conocidas de antemano, tenemos que utilizar una variable para pasárselas a la función.

La variable puntuacion en la función bar( ) no está declarada igual que las variables normales

Realizamos esta tarea con la instrucción:

bar(inpunt);

Ahora cualquier valor que introduzca el usuario será recogido por *scanf( )* y asignado a la variable *inpunt*. Cuando se llama a *bar( )*, se le pasa este valor como argumento. Se han asignado nombres diferentes a las variables: *inpunt* en el programa y *puntuacion* en la función. En realidad, podríamos haber utilizado el mismo nombre para ambas; como se encuentran en diferentes funciones, el compilador las consideraría como variables diferentes.

El argumento en el programa se denomina *argumento actual*, mientras que el argumento de la función invocada es el *argumento formal*. En este ejemplo, la variable *inpunt* constituye el argumento actual del programa, y la variable *puntuacion* es el argumento formal de la función. El conocimiento de estos términos no ayudará a programar mejor, pero no viene nunca de más saber algo de teoría.

BIBLIOGRAFIA RELACIONADA

- *Programación en Microsoft C para IBM PC y compatibles. Introducción y técnicas avanzadas de programación.* Autor: Robert Lafore / Grupo Waite. Anaya Multimedia.
- *Programación en C/C++.* Autor: Chris H. Pappas y William H. Murray. Anaya Multimedia.
- *Aprenda C ya.* Autor: Augie Hansen. Anaya Multimedia. / Microsoft Press.
- *Programación gráfica en C, técnicas avanzadas de modelado y acabado y animación en 3D.* Autor. Lee Adams. Anaya Multimedia.

La pila de acumuladores

La PILA de acumuladores es otro de los conceptos básicos del ensamblador y de la programación en general que es necesario conocer. En este capítulo vamos a mostrar su funcionamiento y las pocas instrucciones relacionadas que posee el 8086 para su programación.

Tras haber explicado ya bastantes conceptos e instrucciones en los anteriores capítulos, como puede ser la forma en que se referencia a la memoria en un 8086, las directivas principales para poder construir el esqueleto de un programa sencillo, las instrucciones más elementales, la definición y uso de procedimientos, etcétera, ahora se va a pasar con la explicación de qué es y cómo funciona la PILA la cual, si nos fijamos bien, ha aparecido en todos los programas de ejemplo que se han dado hasta ahora en el curso, y que como veremos a continuación, es una zona de memoria indispensable tanto para que pueda funcionar el propio procesador de forma correcta, como para que el programador pueda escribir algoritmos complejos dada su utilidad para el almacenamiento de datos temporales cuando las variables a manejar exceden el número de registros disponibles (que no son muchos precisamente en un micro tipo CISC).
También, por su relación indirecta, detallaremos las instrucciones de transferencia de banderas que son, como se podrá ver, muy fáciles de utilizar y comprender.

LA PILA

Hasta ahora, a lo largo de los primeros capítulos del curso se ha referenciado diversas veces acerca de la llamada PILA (*stack* en inglés).
Para empezar decir que se trata de un *búffer* de memoria (*búffer*, segmento, bloque de memoria o como nos guste más llamarle) que se define y usa en todo programa, ya sea de forma automática y transparente al programador, como pasa en el caso de usar lenguajes de alto nivel, o de forma manual, como es el caso del ensamblador, donde disponemos de instrucciones expresas para su manejo.

Con la Directiva STACK definimos un segmento como tipo PILA

Como ya podemos haber observado en los fuentes dados en el curso, para definir un segmento que funcione como PILA sólo tenemos que añadirle la directiva STACK a la de SEGMENT del bloque que queremos definir como tal, (en los listados del capitulo anterior podemos ver ejemplos de su definición).
Otro dato importante es que, como norma general, siempre suele definirse una PILA con un tamaño mínimo de unos 2 Kbytes (2048 bytes), lo cual resulta suficiente para las necesidades de cualquier programa normal que tengamos que desarrollar. Debemos tener presente que en caso de que la pila fuese desbordada en capacidad en alguna ocasión, el fallo de ejecución sería prácticamente seguro y podría dar como resultado un bloqueo de toda la aplicación (*Stack overflow*) o errores secundarios imprevistos de funcionamiento, por lo que reducir su tamaño más de lo normal con el único objetivo de ahorrar memoria se convierte en un riesgo innecesario, dado el pequeño tamaño que requiere para su definición.

PARA QUÉ SE USA LA PILA

En términos generales podemos decir que la PILA se usa para que tanto la CPU como el programador puedan almacenar datos temporales que tienen que ser recuperados poco después.
La característica que más diferencia a la PILA de un bloque típico de memoria es que funciona de forma inversa a como lo hace un *búffer* de memoria normal, o sea, que mientras un bloque típico definido por nosotros lo llenaríamos desde cero hacia el final, incrementando la posición del puntero en la que vamos escribiendo, en la PILA pasa justamente todo lo contrario, se considera el primer dato de la PILA el final *físico* de la misma (el último byte, el offset máxim), y cuando la llenamos de datos completamente llegamos al principio del bloque, donde estaría su posición de memoria relativa 0 (*offset*), quedando claro que cada vez que introduce un elemento, se decrementa el puntero (*Offset*) relativo dentro de la misma.
A esta peculiar forma de funcionar las entradas y salidas de datos de la PILA se la conoce popularmente como LIFO, que son las siglas de Last In, Fisrt Out que se traduce literalmente como *El último en entrar, el primero en salir* y que define perfectamente cómo podemos observar su funcionamiento.
Para complementar la explicación dada, decir que los bloques de memoria normales funcionan del modo llamado FIFO, que son las siglas inglesas First In, First Out y que en español llano sería algo así como *El primero en entrar, el primero en salir.*

Siempre suele definirse una PILA con un tamaño mínimo de unos 2 Kbytes

En la fotografía 1 podemos ver un mapa esquemático del funcionamiento interno de los dos tipos de segmentos que existen en un programa, el modelo FIFO (segmento normal) y el LIFO (PILA), pudiéndose ver todo lo explicado de forma práctica.
Otro aspecto importante de la PILA es que los valores que se almacenan en ella sólo pueden ser elementos de un tamaño mínimo de 2 bytes (palabras), lo que significa que nunca podremos encontrarnos con datos en posiciones que no sean pares respecto al primer elemento que coloquemos en ella, aunque sí podemos almacenar elementos de tamaño múltiple de 2, o sea, elementos de 32 y 64 bits. El porqué de esto no importa demasiado saberlo aún, pero debe tenerse presente puesto que es importante para el buen funcionamiento de la misma. De todas formas, este error no es muy común cometerlo, dado que las propias instrucciones de que dispone el programador para la entrada y salida de datos de PILA no permiten operadores de 8 bits, siendo únicamente posible producirse error si se modifica manualmente el registro SP usado como puntero de pila.

ORÍGENES DEL SISTEMA LIFO

En principio, puede parecer que este sistema de gestión de la pila es un poco innecesario, pero si examinamos los orígenes de los PC's, veremos que su forma de funcionar tenía un por qué.
Cuando los programas funcionaban en los antiguos 8086, era normal que la memoria instalada en el sistema fuese ridícula y que los programas fuesen de pequeño tamaño, del tipo llamado .COM (en vez de EXE), lo cual consistía básicamente en que todo el programa ocupara sólo un segmento de memoria y que, por lo tanto, código, datos y PILA se encuentran en el mismo Segmento (CS por decirlo así). Al ser el espacio disponible tan reducido, y como la PILA no se sabía nunca qué tamaño necesitaría durante la ejecución, se definía un sistema con el cual la PILA empezaba al final del segmento CS y el código y datos empezaban al inicio. De esta forma, se tenía prácticamente todo el espacio de dicho segmento para código y datos, y sólo se llegarían a juntar (sobreescribir) en caso de que la PILA se llenase mucho, situación que ocurriría si se llenase todo el segmento por completo entre datos/código y PILA.

FIFO son las siglas del inglés First In First Out

Otra razón de su forma de funcionar es que, cuando hay parámetros que se pasan por PILA, referenciar a ellos es mas fácil si están en *offsets* positivos que negativos, puesto que, como sabemos, no se pueden usar *offsets* relativos negativos en un puntero de ensamblador.

LA INSTRUCCIÓN 'CALL' Y LA PILA

Como hemos dicho, la PILA es usada por el propio microprocesador para almacenar en ella datos de forma temporal, y tenemos que el principal ejemplo de ello es la instrucción *CALLl*, la cual la aprovecha de la siguiente forma:
Cuando realizamos una llamada a un procedimiento, o sea, cuando ejecutamos un *CALL Procedimiento*, la CPU tiene que disponer de algún medio para poder recordar de alguna forma (almacenar temporalmente en algún sitio) la posición (dirección) de la instrucción siguiente al propio *CALL* para, una vez ejecutado, poder regresar con la ejecución del programa en el lugar donde se dejó (o sea, la instrucción siguiente al *CALL*).
La solución adoptada por los diseñadores de microprocesadores ante este problema fue

COMPARATIVA DEL FUNCIONAMIENTO DE LA MEMORIA FIFO Y LIFO

Mapa interno de un bloque de memoria típico FIFO con elementos tipo word

Contenido	offset
Elemento nº0	0 offset
Elemento nº1	2
Elemento nº2	4
Elemento nº3	6
Elemento nº4	8
Elemento...	10
	12
	14
Elemento...	10600
Elemento 802	10602
Elemento 803	10604
Elemento 804	10606
Elemento 806	10608
Elemento 808	10610
Elemento 809	10612

Mapa de un bloque de memoria tipo PILA (LIFO) con elementos tipo word

Contenido	offset
vacio	0 offset
vacio	2
vacio	4
vacio	6
Elemento 2043	8 — CIMA DE LA PILA ACTUAL (SS:SP)
Elemento 2042	10
Elemento 2041	12
	14
	2034
Elemento...	2036
Elemento nº4	2038
Elemento nº3	2040
Elemento nº2	2042
Elemento nº1	2044
Elemento nº0	2046

ESQUEMA DE ALMACENAMIENTO DE DATOS CON LOS SISTEMAS 'LIFO' Y 'FIFO'.

como podemos deducir ya, la de almacenar la dirección de retorno en la PILA, para luego recuperarla con la instrucción *RET* cuando acabe la ejecución del procedimiento, que pasará el control del programa a dicha dirección.
Como ya se dijo en otro capítulo, los registros usados para apuntar a la PILA son SS:SP, donde SS contiene el segmento que definimos al principio de los programas con la directiva ASSUME y SP es el puntero relativo dentro de dicho segmento.

NUNCA MODIFICAR SS:SP

Es importante que nuestro código nunca altere el contenido de SP o SS, ya que no podemos predecir nunca cuándo el microprocesador puede requerir almacenar o extraer valores de ella, y en caso de que lo hiciese tras haber cambiado nosotros su contenido, se produciría un error fatal. La causa de que sea impredecible no es ningún misterio. Aunque podemos conocer cuándo una instrucción de nuestro programa va a requerir o no acceder a ella, hay un concepto llamado *Interrupciones hardware* que no va a ser explicado ahora, pero que consiste, para que entendamos la situación, en que cuando un periférico envía una señal de aviso a la CPU, como es el caso típico del teclado tras haber pulsado nosotros una tecla, el microprocesador deja la ejecución del programa en curso esté donde esté (o sea, en un lugar impredecible), almacena la posición actual en la PILA (usa SS:SP) y ejecuta una interrupción (rutina) que está asociada a dicho periférico, en este caso, la rutina del DOS que se encarga de capturar y almacenar en un *búffer* la tecla pulsada.

Es importante que nuestro código nunca altere el contenido de SP o SS

Como se puede deducir, si SS:SP no tuviese el valor correcto en el momento de pulsar, por poner un ejemplo el *Enter*, la CPU almacenaría la posición actual de nuestro programa *en un lugar cuya dirección no recuerdo*...para ejecutar después la rutina asociada que quizás también hace uso intensivo de la PILA para almacenar valores temporalmente. El resultado de este ejemplo puede ser cualquier cosa. Un ejemplo simple de entender es que los datos almacenados por la CPU se hubiesen escrito en una zona de memoria donde resulta que está nuestro código de programa, más exactamente, la rutina que dibuja gráficos (*sprites*). Después, cuando nuestro querido videojuego intentase escribir los gráficos de las naves espaciales (por decir algo), quizás llegaría la ejecución de código donde se escribieron bytes basura (datos de PILA), cuyo contenido interpretado como instrucciones podría significar cualquier barbaridad, con el consecuente bloqueo de programa o error de dibujado, en el mejor de los casos.

Autor: José Luis Cr

Tokencraft

Nuestro ganador de este número nos da toda una lección de las posibilidades de DIV. Que no se te da bien dibujar, pues aprovecha los modelos de tus juegos favoritos. Que quieres dar un aire nuevo a los clásicos del hardware lúdico y convertir un juego de estrategia en un arcade, ahora lo tienes fácil.

A estas alturas no creemos que exista alguien que haya comprado esta revista que desconozca las características de *Totenkai* y *Warcraft*; el primero, por ser uno de los primeros juegos comerciales programado con DIV; el segundo, por ser uno de los títulos de estrategia que marcaron toda una época.
El autor de *Totencraft* ha dado un homenaje a estos dos juegos mezclando los elementos de los escenarios y los personajes de *Warcratf* con la acción y la jugabilidad de *Totenkai*.
Resultado: un trepidante arcade en el que tendremos que acabar con los orcos de *Warcraft*, pero no a base de devanarnos los sesos, si no a fuerza de tirar de gatillo.
Totencraft muestra una forma fácil y divertida de construir juegos sin que tengamos que recurrir a un costoso trabajo de diseño de personajes y escenarios. No hay que olvidar que juegos de éxito tienen liberado su código para todo aquel que lo quiera bajar de Internet. *Descent 1* y *2*, *Quake* y muchos más pueden convertirse en una fuente inagotable de elementos que podemos aprovechar para realizar nuestros propios trabajos.
Claro que, una vez ejercitados, deberíamos probar a desarrollar un juego con miras comerciales, pero para ello hoy en día es casi imprescindible contar con un buen grupo de diseñadores 2D, 3D, guionistas, programadores y demás fauna de esta selva del software lúdico.
Pero pasemos a presentar este juego.

UN HOMENAJE A DOS CLASICOS

La acción se desarrolla en una comuna orca. ¿Qué diablos pinta el protagonista de *Tokenkai* en un centro de rehabilitación de antiguos combatientes orcos?
Mira lo que dicen los orcos. El protagonista no tiene más que una metralleta y varias





granadas, ¿qué puede hacer? Fíjate en las reacciones de los orcos antes y después de atacarles.
La metralleta funciona mucho mejor que las granadas, éstas sólo funcionan de vez en cuando.
Botón izquierdo del ratón: moverse.
Botón derecho del ratón: dispara el arma seleccionada.

Puedes seleccionar el arma pulsando en los iconos de la barra inferior, o pulsando:
1 - para la metralleta.
2 - para las granadas.

La tecla asterisco (*) del teclado numérico si para hacer copias de pantalla, no se te olvid

Manejo del editor de mapas

Botón izquierdo del ratón: coloca la baldosa seleccionada. Botón derecho del ratón: quit un orco.
Las casillas se seleccionan pulsando sobre el botón correspondiente de la barra superior. barra de botones cambia cuando pulsas el botón gris.
F5: grabar mapa.
F6: cargar mapa.
ESC: salir.
Bugs: las granadas funcionan a veces, y otras

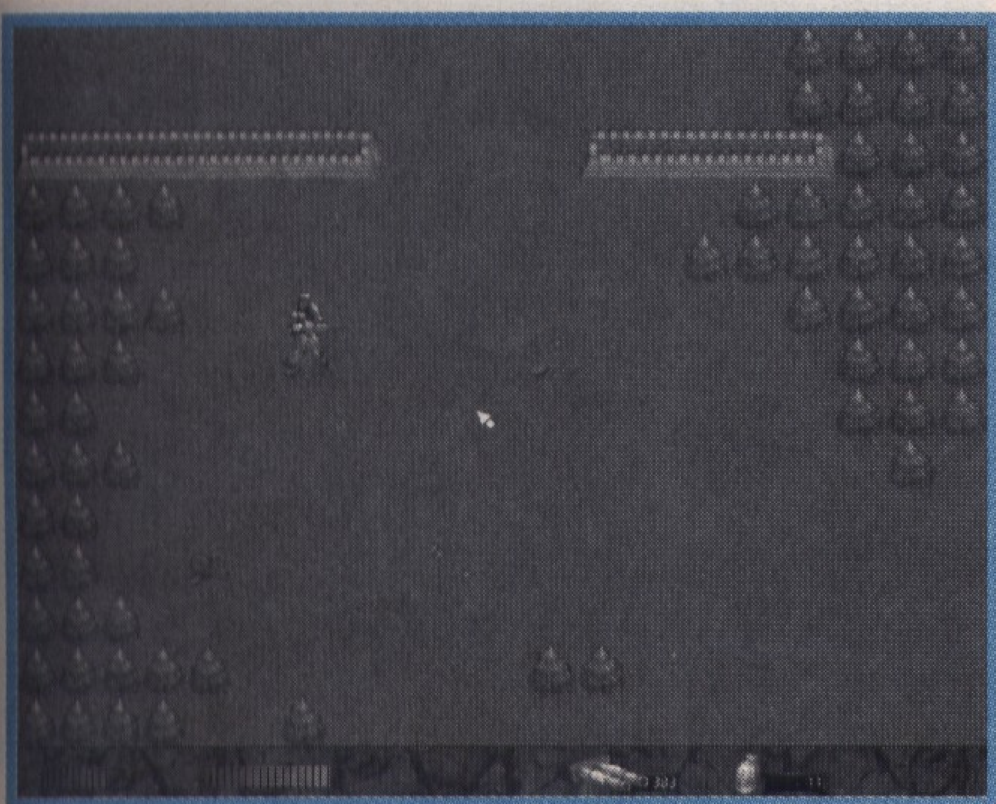
Código

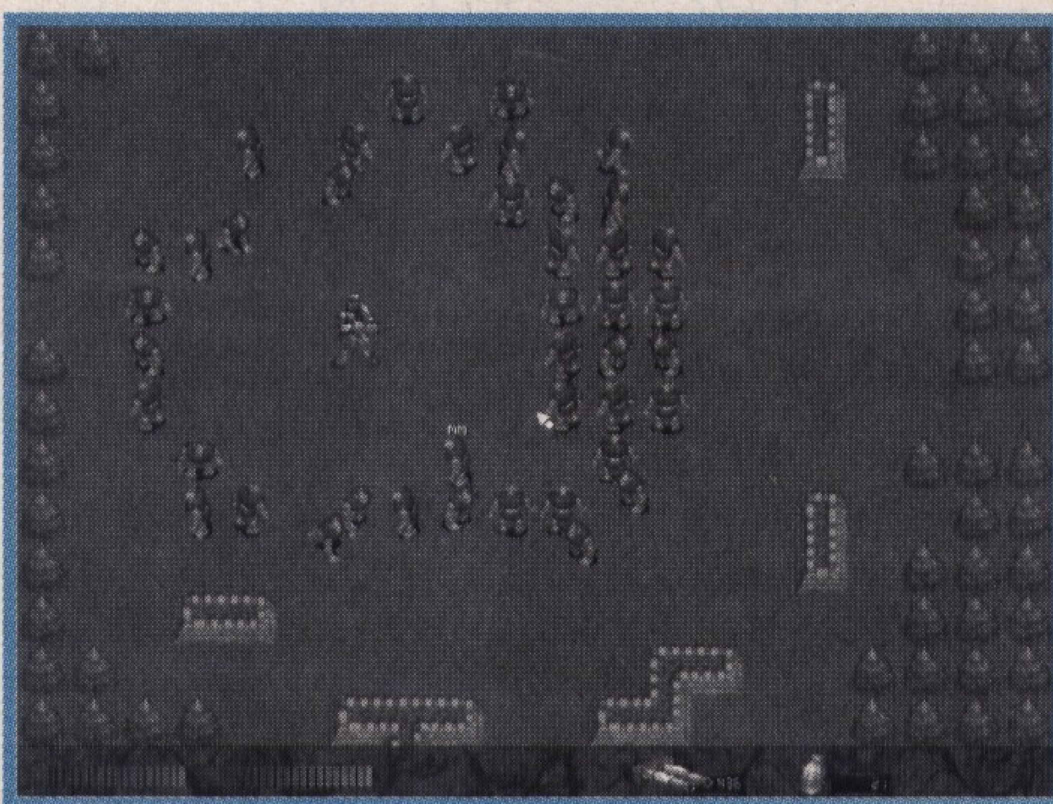
```
compiler_options
  _extended_conditions, _case_sensitive,
_max_process=200;

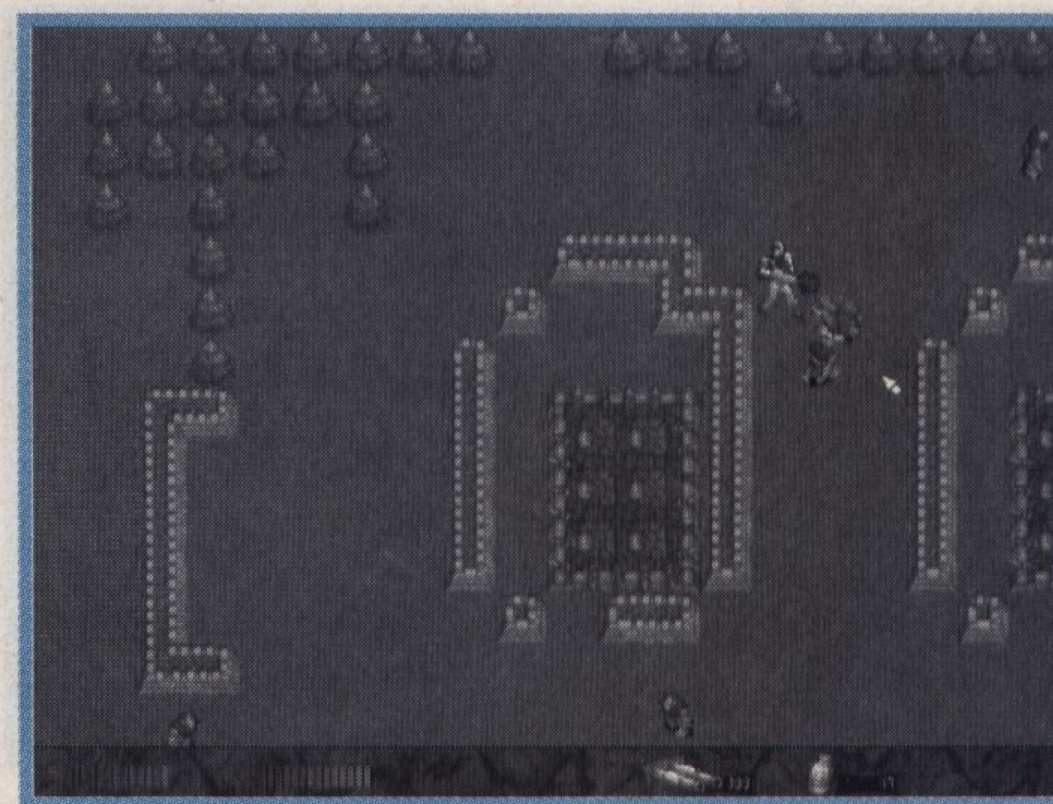
program tokencraft;

const
  // TABLERO
  filas_tablero=48;
  columnas_tablero=48;
  filas_visibles=14;
  columnas_visibles=20;
  alto_baldosa=320;
  ancho_baldosa=320;
```









```
    max_puntos=20;              // m ximo de puntos
para la b£squeda de caminos de ego
    max_habitantes=50;         // n£mero m ximo
de habitantes
    num_destinos_habitante=20; // m ximo de
puntos para la b£squeda de caminos de los
enemigos

    // constates de los enemigos
    visi¢n_pe¢n=8;
    pausa_ataque_pe¢n=4;
    dist_atacar_pe¢n=2;
    dist_aviso=5;
    // da¤o de las armas
    espada=         10;
    metralleta=     20;

    // interfaz
    mouse_scroll=3;
    // botones
    x_men£_princ=320;
    y_men£_princ=250;
    // juego
    comienzo_armas_x=400;
    comienzo_armas_y=414;
    // cargar mapa
    x_cargar=160;
    y_cargar=112;
    // fin de juego
    jg_x=140;
    jg_y=130;
    jg_txt_x=330;
    jg_txt_y=110;
    jg_txt_esp=20;

    // duraci¢n del texto de los enemigos
    tiempo_max_texto=100;


global
    ia_orcos;  // 0 - orcos amistosos
               // 1 - orcos en guerra


    mapas_textos[15+16+16+8];  // donde se
guardan los mapas de los textos
    textos_orcos[15+16+16+8]=  // lo que dicen
los orcos, se guarda en mapas_textos[]
        // sonidos 16
        "!","úúú","uhm","orc","om","¨","?","!",
        "","¨?","úúú","ay","uff","ou","mm","ss",
        // nivel 0 16
        "hola","me llamo Orc","quien es","que
raro","yup","jope","miralo",
        "es nuevo","¨viste a ese?","mira
ese","hala","jo","fijate","ya
veo","mira","anda",
        // nivel 1 - atacar 16
        "VETE","FUERA","ASESINO","MATON","TE
VAYAS","SAYONARA","TONTOLABA","BOLUDO
",

"asasinos","AAAA","AAAAH","AAA","YAAAR","O
OORC","muerte","guerra",
        // nivel 1 - huir 8

"socorro","auxilio","ayuda","mama","nooo","a
aaah","ay ay","ay dios";

    // procesos
    id_ego;

    // ia
    map_obst culos;
    map_copia_pantalla;

    // mapa de juego
    mapa_elegido;

    // lista de destinos
    struct destinos_ego[30] x,y; end
    struct destinos_habitantes[max_habitantes]
        struct
destinos[num_destinos_habitante+10] x,y; end
        ocupado;
    end

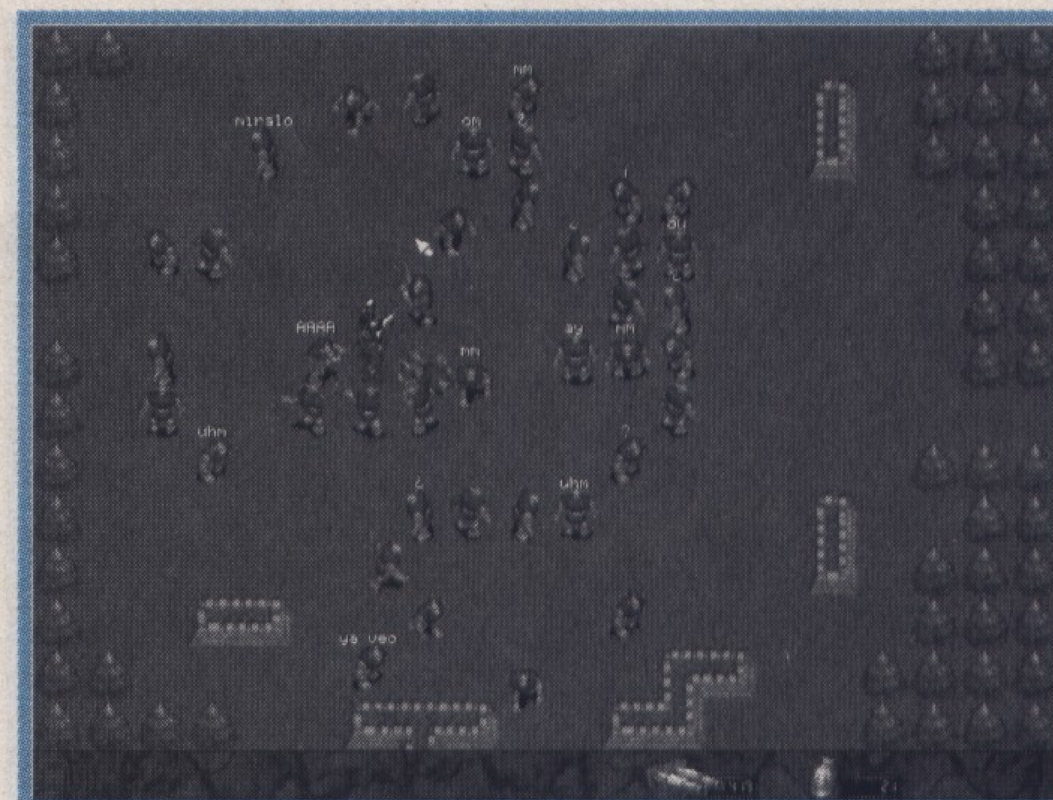
    // coordenadas en el mapa
    ego_x_casilla, ego_y_casilla;
    x_casilla_cursor, y_casilla_cursor;

    // identificadores de datos

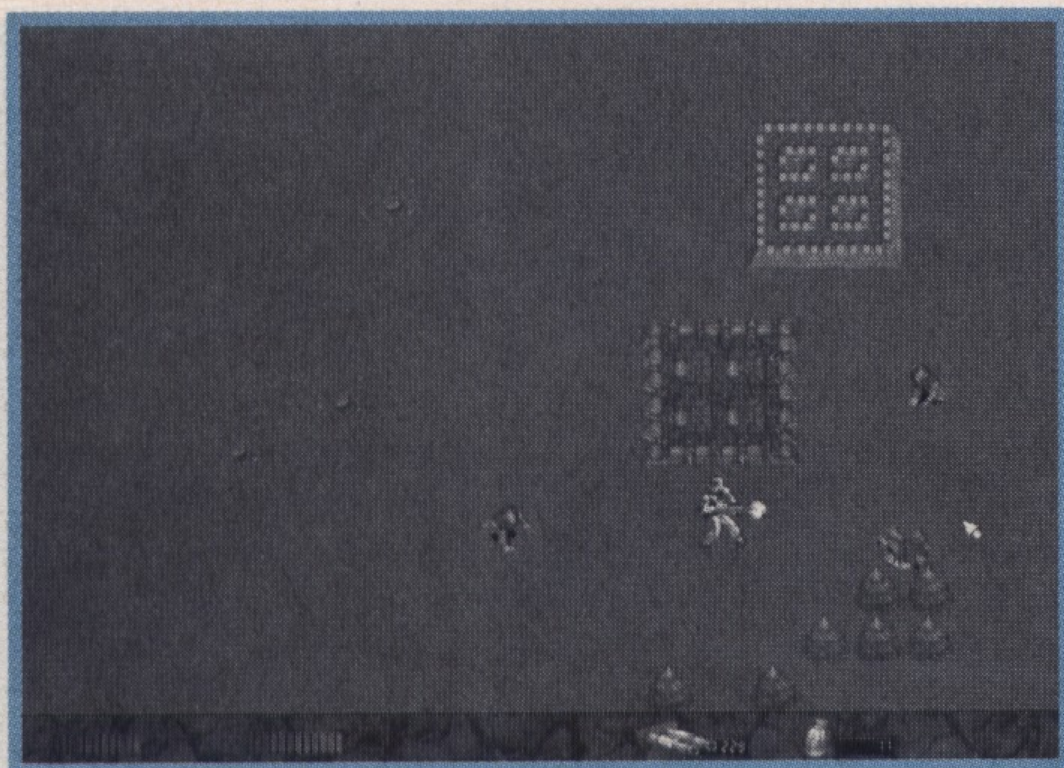
    // fnt
    fuente[2];
    // fpg
    fichero[5];
    // mod
    m£sica;
    // wav
    sonido[25];


    // refencias del tablero
    fila_superior; columna_izquierda;
    fila_graph2; columna_graph2;


    // ANIMACIONES EGO
    anima[]=
        16,1,09,17,25,33,-25,-17,-09,-1,-65,-57,-
49,41,49,57,65,
        16,2,10,18,26,34,-26,-18,-10,-2,-66,-58,-
50,42,50,58,66,
        16,3,11,19,27,35,-27,-19,-11,-3,-67,-59,-
51,43,51,59,67,
        16,4,12,20,28,36,-28,-20,-12,-4,-68,-60,-
52,44,52,60,68,
        16,5,13,21,29,37,-29,-21,-13,-5,-69,-61,-
53,45,53,61,69,
        16,6,14,22,30,38,-30,-22,-14,-6,-70,-62,-
54,46,54,62,70,
        16,7,15,23,31,39,-31,-23,-15,-7,-71,-63,-
55,47,55,63,71,
        16,8,16,24,32,40,-32,-24,-16,-8,-72,-64,-
56,48,56,64,72;
    prodis[]=
        16,73,76,79,82,85,-82,-79,-76,-73,-97,-
94,-91,88,91,94,97,
```



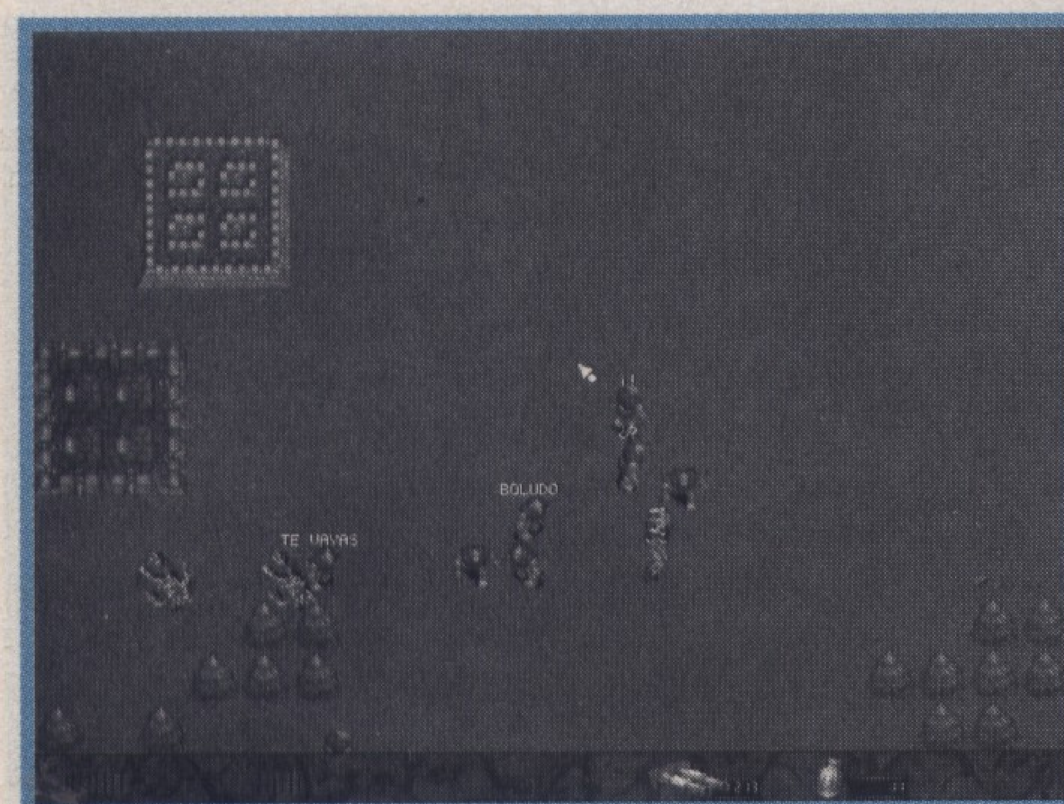
Juegos ganadores: 1º



```
        16,74,77,80,83,86,-83,-80,-77,-74,-98,-
95,-92,89,92,95,98,
        16,75,78,81,84,87,-84,-81,-78,-75,-99,-
96,-93,90,93,96,99;

   // ANIMACIONES ENEMIGOS
   andar[]=
        8,1,11,21,-11,-1,-41,31,41,
        8,2,12,22,-12,-2,-42,32,42,
        8,3,13,23,-13,-3,-43,33,43,
        8,1,11,21,-11,-1,-41,31,41,
        8,4,14,24,-14,-4,-44,34,44,
        8,5,15,25,-15,-5,-45,35,45;
   atacar[]=
        8,06,16,26,-16,-06,-46,36,46,
        8,07,17,27,-17,-07,-47,37,47,
        8,08,18,28,-18,-08,-48,38,48,
        8,09,19,29,-19,-09,-49,39,49,
        8,10,20,30,-20,-10,-50,40,50;
   morir[]=
        8,51,51,51,-51,-51,-54,54,54,
        8,52,52,52,-52,-52,-55,55,55,
        8,53,53,53,-53,-53,-56,56,56;
   cuerpos[]=
        8,60,60,60,-60,-60,-64,64,64,
        8,61,61,61,-61,-61,-65,65,65,
        8,62,62,62,-62,-62,-66,66,66,
        8,63,63,63,-63,-63,-67,67,67;


   struct inicio  // Datos para iniciar una partida
      energ¡a;   // energ¡a (de 0 a 90)
      escudo;    // Escudo (de 0 a 90)
      balas_me;  // Munici¢n de metralleta
      granadas;  // Granadas
   end
```



```
   arma_sel;   // arma seleccionada por ego
   max_arma[]=350,25;  // Munici¢n m xima
para cada arma

   id_textos[12];  // identificadores de los textos
   string map_elegido[11]; // mapa elegido para
cargar
   string num_scr[11];     // para grabar las
capturas de pantalla


   total_enemigos;   // cu ntos enemigos hay
vivos
   enemigos_10; // cu ntos hay atac ndole
   enemigos_9;  // cu ntos hay persigui,ndole


   // TABLERO
   //——————————————
   struct tablero[columnas_tablero, filas_tablero];
      base;
      terreno;
      habitantes;
   end


local
   x_casilla, y_casilla; // coordenadas en la
cuadr¡cula del mapa
   estado;
   texto;
   energ¡a;
   escudo;

private
   n, nc, nf;  // contadores de columna y de l¡nea,
otros

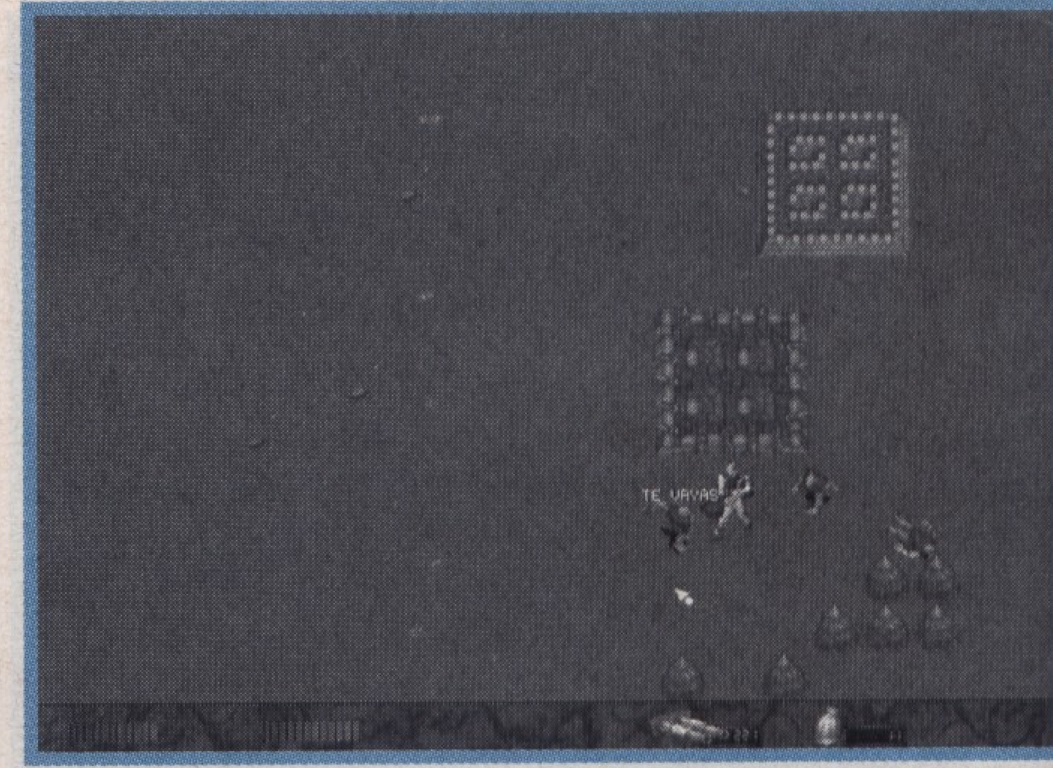
begin
   set_mode(m640x480);
   set_fps(33,3);

   // aviso de carga

   write(0,260,330,0,"cargando datos...");
   uncompress_file("fpg\imagen.fpg");

   fichero[5]=
load_fpg("tcraft\imagen.fpg");
   put(fichero[5],100,300,130);
   compress_file("fpg\imagen.fpg");
   unload_fpg(fichero[5]);
   frame;


   from n=1 to 200;
      frame;
   end
    /*************
      CARGA DATOS
    *************/
```



```
   /* ***** PAL ***** */
   force_pal("tcraft\paleta.pal");

   /* ***** FNT ***** */
   fuente[0]=
load_fnt("tcraft\fuente2.fnt");
   fuente[1]=
load_fnt("tcraft\fuente1.fnt");
   fuente[2]=
load_fnt("tcraft\fuente3.fnt");

   /* ***** FPG ***** */
   uncompress_file("fpg\terreno.fpg");
   uncompress_file("fpg\misc.fpg");
   uncompress_file("fpg\ego.fpg");
   uncompress_file("fpg\peon.fpg");
   uncompress_file("fpg\imagen.fpg");
   fichero[0]=
load_fpg("tcraft\terreno.fpg");
   fichero[1]=
load_fpg("tcraft\interfaz.fpg");
   fichero[2]=     load_fpg("tcraft\misc.fpg");
   fichero[3]=     load_fpg("tcraft\ego.fpg");
   fichero[4]=     load_fpg("tcraft\peon.fpg");
   fichero[5]=
load_fpg("tcraft\imagen.fpg");
   compress_file("fpg\imagen.fpg");
   compress_file("fpg\terreno.fpg");
   compress_file("fpg\misc.fpg");
   compress_file("fpg\ego.fpg");
   compress_file("fpg\peon.fpg");


   /* ***** MOD ***** */
   uncompress_file("mod\fuente.mod");
   m£sica=load_song("tcraft\fuente.mod",1);
   compress_file("mod\fuente.mod");

   /* ***** WAV ***** */
   sonido[0]=
load_wav("tcraft\impacto0.wav" ,0);
   sonido[1]=
load_wav("tcraft\impacto1.wav" ,0);
   sonido[2]=
load_wav("tcraft\impacto2.wav" ,0);
   sonido[3]=
load_wav("tcraft\impacto3.wav" ,0);
   sonido[4]=
```

```
load_wav("tcraft\impacto4.wav" ,0);
  sonido[5]=
load_wav("tcraft\impacto5.wav" ,0);
  sonido[6]=
load_wav("tcraft\impacto6.wav" ,0);
  sonido[7]=
load_wav("tcraft\nopuedo.wav" ,0);
  sonido[8]=
load_wav("tcraft\pistola.wav" ,0);
  sonido[9]=
load_wav("tcraft\giro07.wav" ,0);
  sonido[10]=
load_wav("tcraft\impacto.wav" ,0);
  sonido[11]=
load_wav("tcraft\boton00.wav" ,0);
  sonido[12]=
load_wav("tcraft\explo1.wav" ,0);
  sonido[13]=
load_wav("tcraft\pasos.wav" ,0);
  sonido[14]=
load_wav("tcraft\morir.wav" ,0);
  sonido[15]=        load_wav("tcraft\click.wav"
,0);
  sonido[20]=
load_wav("tcraft\espada1.wav" ,0);
  sonido[21]=
load_wav("tcraft\espada2.wav" ,0);
  sonido[22]=
load_wav("tcraft\espada3.wav" ,0);
  sonido[23]=
load_wav("tcraft\grito1.wav" ,0);
  sonido[24]=
load_wav("tcraft\grito2.wav" ,0);
  sonido[25]=
load_wav("tcraft\grito3.wav" ,0);

  // crea los gr ficos de los textos
  from n=0 to 15+16+16+8;

mapas_textos[n]=write_in_map(0,textos_orcos[n
],4);
  end

  delete_text(all_text);

  // men£
  men£_principal();
end

//————————————

process men£_principal();

private
  n;
  byte pulsado[2];

begin
  // inicializa
  let_me_alone();
  clear_screen();
  delete_text(all_text);
  delete_draw(all_drawing);
  if (!is_playing_song()) song(m£sica); end
  fade_on();

  // fondo
  put_screen(fichero[5],900);
  // cursor
  mouse.file=fichero[0];
  mouse.graph=997;
  // botones
  bot¢n(x_men£_princ,y_men£_princ,   1,
"Jugar Escenario");
  bot¢n(x_men£_princ,y_men£_princ+40,1,
"Opciones");

bot¢n(x_men£_princ,y_men£_princ+100,1,"Salir
");
  // texto
  write(0,10,460,0,"TOKENCRAFT  beta 0.8 es.
Copyright (C) 1999 Jos, Luis Cruz");

  loop
    // jugar
    if ((pulsado[2] and !key(_enter))
      or
(mouse_in(x_men£_princ,y_men£_princ,x_men£
_princ+224,y_men£_princ+28) and pulsado[1]
and !mouse.left))
        cargar_mapa(); break;
    end

    // cargar escenario
    if
(mouse_in(x_men£_princ,y_men£_princ+40,x_m
en£_princ+224,y_men£_princ+40+28) and
pulsado[1] and !mouse.left)
        opciones(); break;
    end

    // salir
    if ((pulsado[0] and !key(_esc))
      or
(mouse_in(x_men£_princ,y_men£_princ+100,x_
men£_princ+224,y_men£_princ+100+28) and
pulsado[1] and !mouse.left))
        delete_text(all_text);
        let_me_alone();
        break;
    end
    pulsado[0]=key(_esc);
    pulsado[1]=mouse.left;
    pulsado[2]=key(_enter);
    frame;
  end
end

//————————————————————————
```




```
//————————————————————————
// Proceso de control del juego
//————————————————————————

process juego();

private
  fila_tabla; columna_tabla;
  fila_graph; columna_graph;
  //
  idn;

  n, nc, nf;
  pulsado[2];

begin
  // inicializa y limpia
  delete_text(all_text);
  delete_draw(all_drawing);
  clear_screen();
  let_me_alone();
  frame;
  resolution=10;
  ctype=c_scroll;
  // inicializa
  fila_graph2=0; columna_graph2=0;
  fila_superior=0; columna_izquierda=0;
  // limpia el mapa de fondo
  from nf=0 to filas_visibles;
    from nc=0 to columnas_visibles;

map_put(fichero[0],990,2,nc*ancho_baldosa/10,
nf*alto_baldosa/10);
    end
```



Autor: Javier Arche Canales

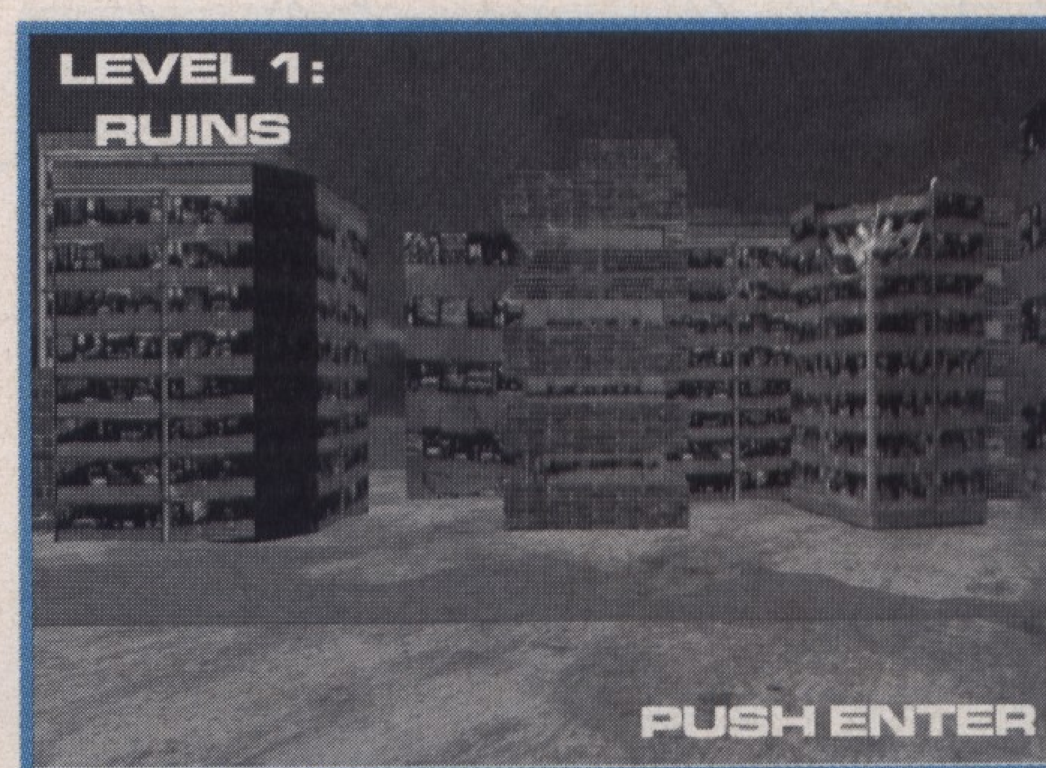
Nowadays

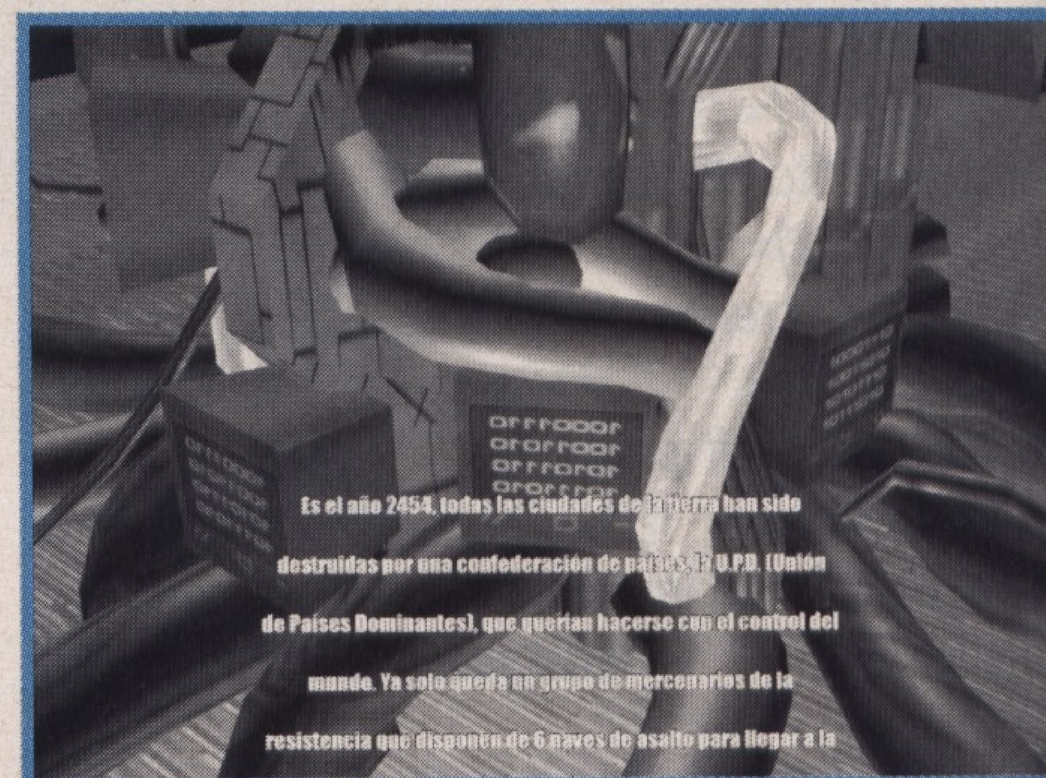
Un juego que evoca a las antiguas máquinas recreativas del género arcade. Os ponemos en situación: hay que manejar un avión que encontrará los cielos plagados de enemigos. Menos mal que nuestra pequeña nave es capaz de disparar sin cesar y limpiar la atmósfera de todos los contrincantes que encontremos

En este juego no hay que pensar mucho. Consiste en manejar un pequeño avión y darle gusto al dedo. Hay varios tipos de armas que iremos recogiendo a medida que avancemos, un escudo que nos dará invulnerabilidad por un tiempo y una miríada de enemigos con los que probar nuestra habilidad. Si algún avezado jugador tiene dificultades para pasar el nivel siempre se puede echar mano de los trucos que nos proporciona el autor del juego. Pero démosle ya la palabra, o mejor su espacio en el papel.

EL AUTOR Y SU OBRA

Saludos hermanos, soy el creador del juego que habéis jugado o estáis a punto de jugar.
Lo que está dentro de vuestra unidad CD-ROM son 8 duros meses de trabajo, Nowadays, un juego del que se han visto pantallas en multitud de sitios. En Internet (en las paginas de StRAtOS, Danger Pictures o Divnet) y en la revista Divmanía.
Mucha gente estaba deseando probar el juego sin gastarse un duro y finalmente lo han conseguido. Yo ya me he cansado de él porque han aparecido juegos mejores hechos en Div y porque las distribuidoras no le veían ningún futuro. Si el juego gana algún premio en la revista os prometo que el dinero no será malgastado sino que ayudará a superar este pequeño tropiezo y ayudará al futuro de la programación española.
Con todo esto no quiero decir que el juego sea lo mejor que hay en Div, habrá gente a la que le guste y gente a la que no, pero hay está el juego.
Quiero darles mis agradecimientos a la gente de StRAtOS, Atapuerca, Pirro, Micro, Manowar, etc. que me han ayudado mucho, a mis colegas y a la gente que se dedique a los videojuegos, ya sea programándolos o diseñándolos o cualquier cosa que forme un videojuego. Ah, y también a todos los DJ Trackers y Dj´s nacionales.
Además a la chica que dicen que hay por ahí programando en Div, le digo que siga así y que es una tía cojonuda, aunque nunca he hablado con ella. A ver si toman ejemplo muchas mujeres y podemos decir que el chiste que hay en la pagina de StRAtOS de "por qué los coders nunca tendrán novia" es absurdo.



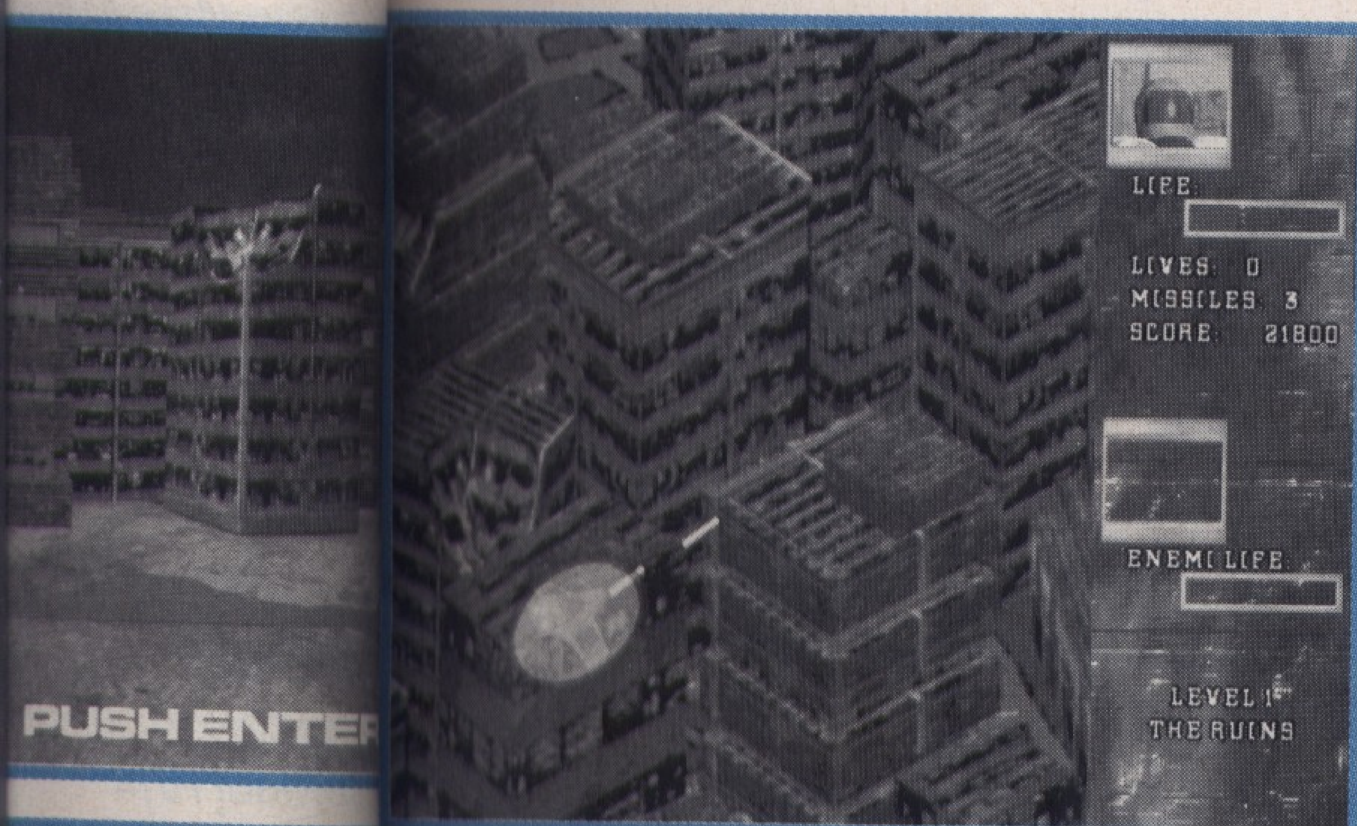


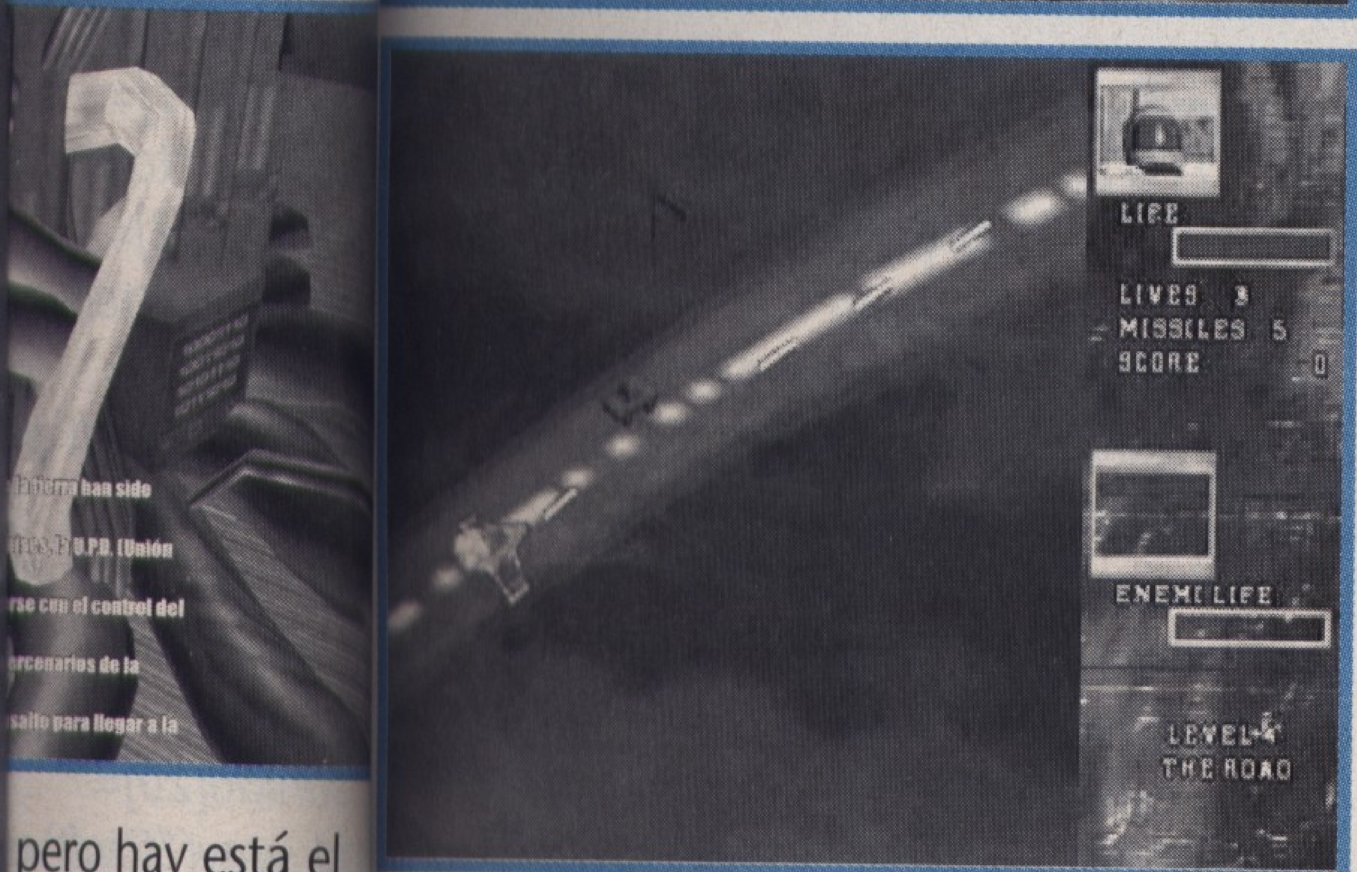
EL JUEGO

Este es mi primer juego. Ya sé que no se debe empezar por hacer un juego comercial, pero es que me hacía ilusión.
Primero hice los bocetos de las naves y los fui pasando a 3D junto con los escenarios. El argumento lo hice en una tarde en plan cutre y la mayoría de las ideas se me iban ocurriendo sobre la marcha por lo que el juego no se planeó mucho.
El código fuente, como se puede ver, no esta terminado. Pensaba revisarlo todo al final para optimizarlo.

Se despide:
MaSTeR RhaPSoDY (Javier Arche Canales)
Danger Pictures.
Diox creó el universo, yo lo pasé a baja poligonización.









TRUCOS

Por si no sois capaces de pasaros el juego, os pongo aquí los trucos para que os ayuden un poco:

Nowa0- Pone la vida a 999999.
Nowa1- Te lleva al siguiente nivel.
Nowa2- Reinicia el nivel actual con 3 phantoms y el 2º disparo.
Nowa3- Reinicia el nivel actual con 3 phantoms y el tercer disparo.
Nowa4- Nos da el 2º disparo.
Nowa5- Nos da el tercer disparo.
Nowa6- Disparo láser.
Nowa7- Nos da el escudo.
Nowa8- Reinicia el nivel actual con 3 phantoms y el primer disparo.
Nowa9- Reinicia el nivel actual y nos deja como al principio, sin trucos.
L- al pulsar esta tecla se borran los trucos, por si nos hemos equivocado al escribirlos.

CODIGO FUENTE DEL JUEGO

```
//————————————————————————————————
//        NOWADAYS VERSION 1.0 (C) 1999 DANGER PICTURES
//        CREADO POR JAVIER ARCHE CANALES (ALIAS RaTHSoDiC)
//        EN JUNIO DEL 99. E-MAIL: RATH@LETTERA.NET
//        C/ LUIS DORADO N§56 C.P.13500 PUERTOLLANO (C.REAL)
//        SI QUIERES SABER COMO CONSEGUIR EL JUEGO COMPLETO ENVIAME
//        UN E-MAIL PARA PREGUNTAR, PORQUE SEGURAMENTE EL JUEGO
//        NO SEA DISTRIBUIDO.
// HTTP://www.geocities.com/SiliconValley/Bit/9451
//————————————————————————————————

compiler_options
_ignore_errors;

program NowadayS;

global

  //musicas

musica_tension,musica_title,song_selec,musica_fase1;

  //sonidos

explosion_prota,sonido_title,explosion_enemi,sonido_laser,sonido_logo;
  sonido_cursor,bonus_sound,colision_s;

  //fuentes
  fuente_fase1;

  //mapas
  mapa_present,map_level, fondo_intro;

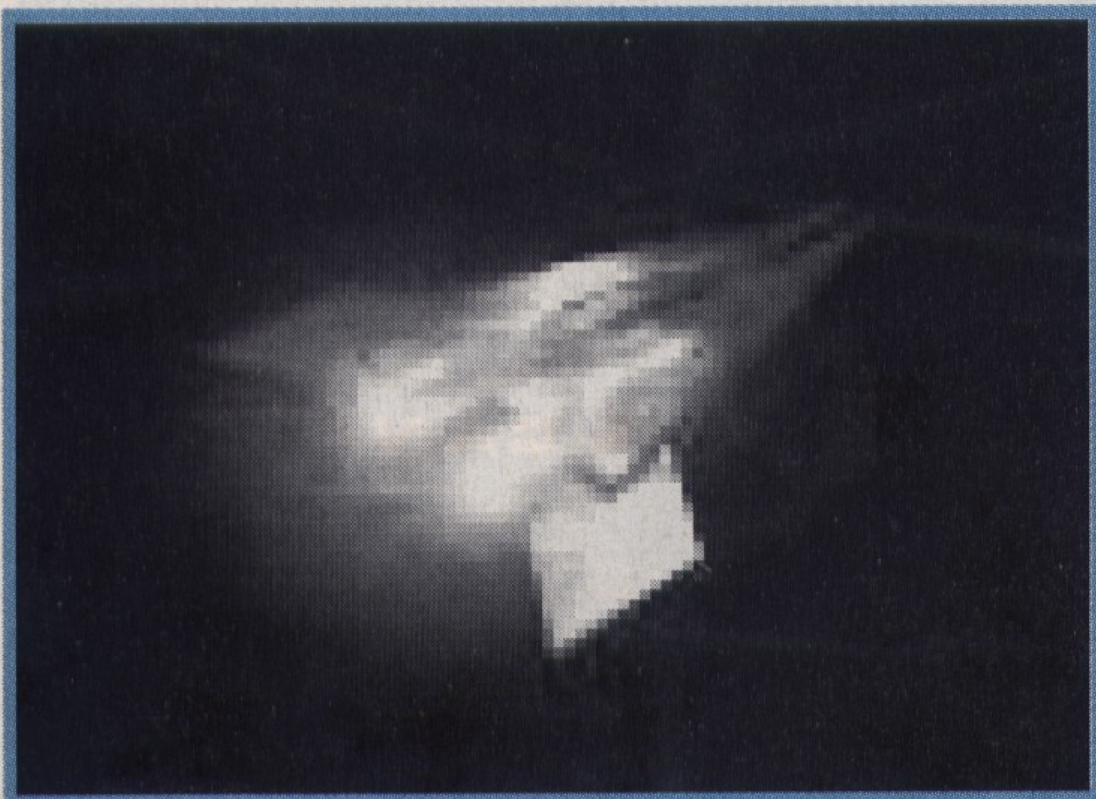
  //ficheros

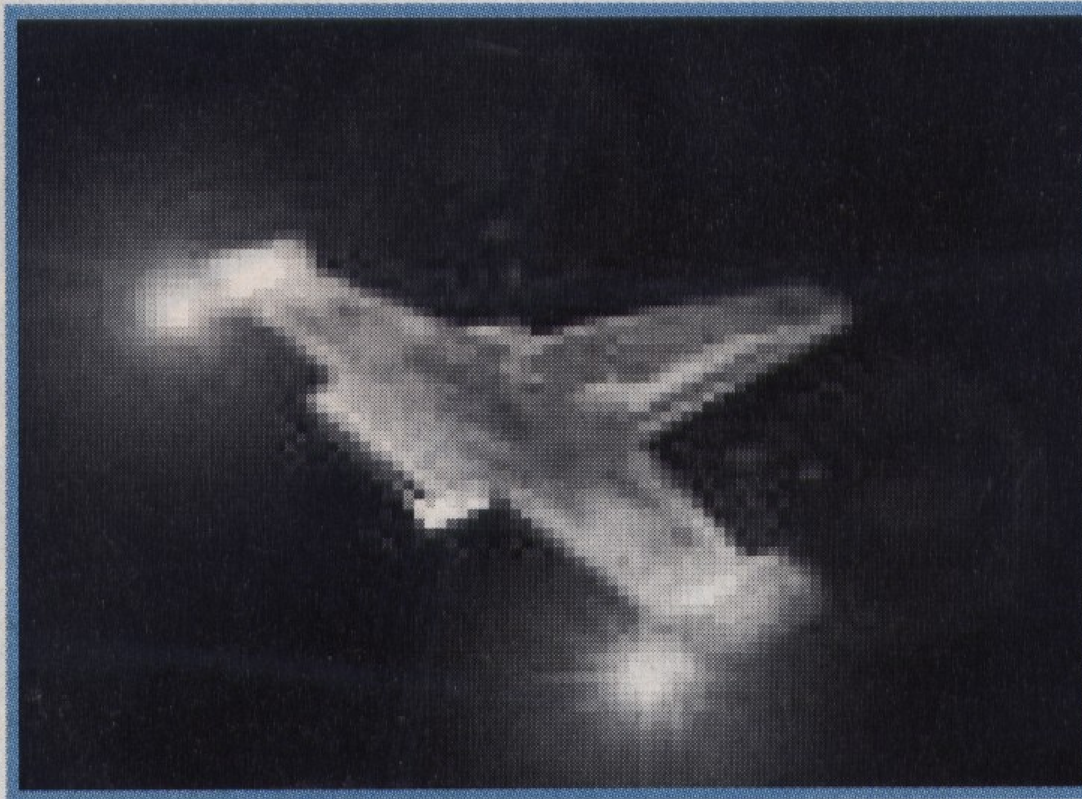
map_logo,fpg_selec,fpg_title,fpg_nave1,fpg_marc,fpg_mesages,fpg_bonus,ffinal;

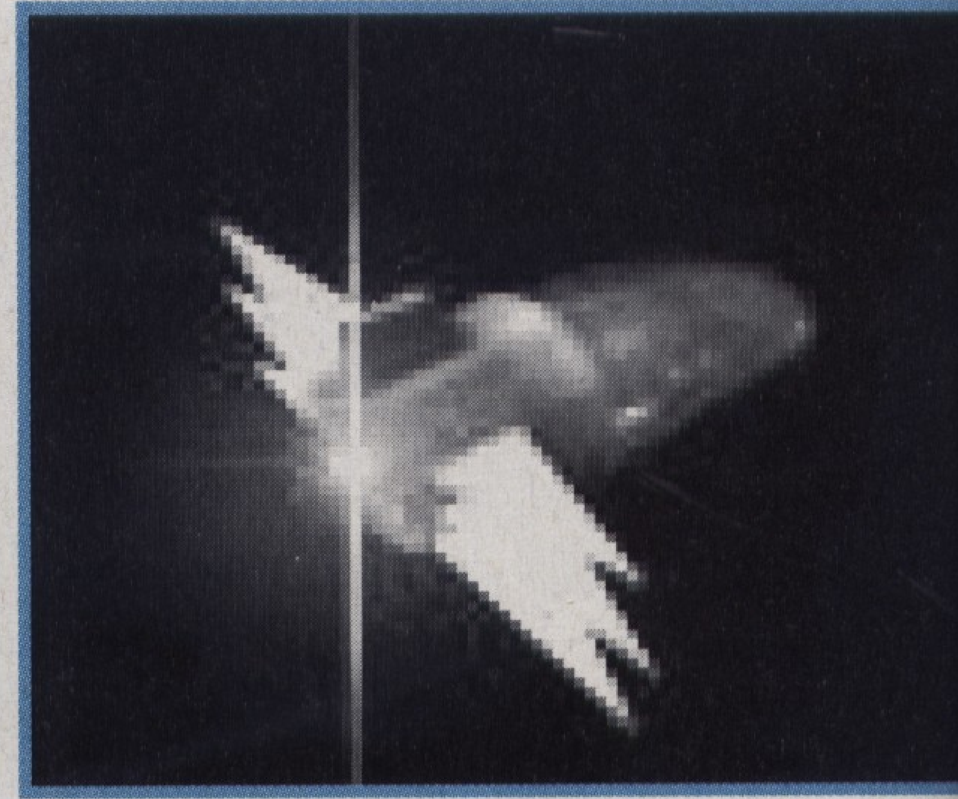
fichero_gover,fpg_enemis,fpg_ds_enemi,fpg_ds_prot,fpg_f_n,fpg_explos,fpg_extras;
    //variables
    posicioncursor=0;
    id_programa_principal;
    opcion_menu=-1;
    posicionselec=0;
    opcion_nave1=1;
    id_selec_nave;
    nave_elegida=1;

    misil_presente=0;
    id_nave1;
    valor_bonus=1;
    escudo_presente;
    x_nave;
    y_nave;
    z_nave;
    an_nave;
    enemigos_tipo1_muertos=0;
    enemigos_tipo2_muertos=0;
    phantom_right=0;
    phantom_left=0;

scroll_x;
scroll_y;
nave_parado[]=8,8,8,9,9,9,10,10,10,11,11,11,12,12;
nave_energia=190;
max_vidas=9;
vidas=3;
misiles=6;
```







```
lasers_lanzados=0;
lasers_lanzados2=0;
lasers_lanzados3=0;
nivel_disparo=1;
puntos=0;
velocidad_nave=6;
velocidad_max=9;
fuerza_disparo=10;
fase=1;

fondo_credits,fuente_credits1,fuente_credits2;
letra_intro, musica_intro;

local
valor_x_actual;
valor_y_actual;
resistencia;

begin
set_mode(m640X480);          //pone el
modo de pantalla
load_pal("c:\proyec~1\estatica\logo.pcx");
//carga la paleta
MAP_logo=load_pcx("c:\proyec~1\estatica\logo.
pcx"); //carga el fichero
put_screen(0,map_logo);          //pone el
grafico en pantalla
timer[0]=0; //pone el crono 1 a 0
while (timer[0]<500) frame; end       //deja el
grafico en pantalla 5 seg
intro();  //pasa al proceso intro
end

//——————————————————————————————

// PROCESO INTRO
//——————————————————————————————

PROCESS INTRO();
BEGIN
set_fps(20,0);
let_me_alone();
unload_pcx(map_logo);
unload_wav(sonido_logo);
clear_screen();
load_pal("c:\proyec~1\estatica\fondo.pcx");
fondo_intro=load_pcx("c:\proyec~1\estatica\fond
o.pcx");
letra_intro=load_map("c:\proyec~1\estatica\intr
o.map");
musica_intro=load_song("c:\proyec~1\musica\in
tro.xm",1);
put_screen(0,fondo_intro);
timer[0]=0;
graph=letra_intro;
song(musica_intro);
x=320;
y=750;
loop
y—;
if(scan_code==1)
title();
break;
end
if(y<240)
y=240;
end
if(timer[0]>2900)
title();
break;
end
frame;
end
end

//——————————————————————————————

// PROCESO TITLE
// CREA LA PANTALLA DE PRESENTACION
//——————————————————————————————

process title();
begin
   set_fps(30,0);
   let_me_alone();
   unload_pcx(fondo_intro);     //descarga el
fichero
   clear_screen();
   opcion_menu=-1;
   posicioncursor=0;

musica_title=load_song("c:\proyec~1\musica\no
wada.xm",1);

sonido_cursor=load_pcm("c:\lazer\sonido\pi
av",0);    //carga sonido
   load_pal("c:\proyec~1\fpg\text.fpg");
//carga paleta
   id_programa_principal=id;

fpg_title=load_fpg("c:\proyec~1\fpg\text.fpg
// carga fichero
   song(musica_title);
   letras();
   selec();
   timer[1]=0;
   loop
      if(timer[1]>6000)
      clear_screen();
      stop_song();
      unload_song(musica_title);
      unload_wav(sonido_cursor);
      unload_fpg(fpg_title);
      intro();
      break;
      end

      if (scan_code==_down)
      sound(sonido_cursor,256,256);
      posicioncursor ++;
         if (posicioncursor>2)
            posicioncursor=0;
         END
      END
      if (scan_code==_up)
      sound(sonido_cursor,256,256);
      posicioncursor —;
         if (posicioncursor<0)
            posicioncursor=2;
         END
      END
      if (scan_code==_enter)
      opcion_menu=posicioncursor;
      END
      if(opcion_menu<>-1)
         SWITCH (opcion_menu)
            case 0:
            selec_ship();
            break;
```

Autor: Enrique López Martos

JUEGOS GANADORES: 3º

Alex Exoddus



El tercer premio ha recaído en este juego de plataformas, estilo *Mario Bros*, que destaca por su gran dificultad. Tiene varios niveles que habrá que recorrer hasta llegar al final del juego y habrá que sortear los disparos de los múltiples enemigos que nos saldrán al paso.

Como os decimos, es bastante difícil progresar a través de *Alex Exoddus*. La razón es que nuestras municiones, bombas arrojadizas, son limitadas. Debemos usarlas para acabar con todos los obstáculos que encontramos en el camino, en forma de dragones escupe-fuego por ejemplo.
Los enemigos, que no paran de moverse, son difíciles de alcanzar. Si erramos varios tiros y se nos agota la munición tendremos bastante crudo pasar al nivel siguiente. Pero este alto grado de dificultad le suma adicción al juego y no se la resta en ningún caso. Así que a saltar y a tirar bombas con mucho tino.

INFORMACION

Las teclas que es necesario utilizar en *Alex Exoddus* son las siguientes:
Mover: cursor izquierda/derecha.
Salto: cursor arriba.
Disparo: espacio.
Salir: esc.

Código

```
//————————————————————————
// TITULO:  ALEX EXODDUS
// VERSION: 0.1
// AUTOR:   MARTOS GAMES
// FECHA:   05-11/01/2000
```

ALEX EXODDUS
Izquierda <
Derecha >
saltar ^
Bomba space
Salir esc

```
//————————————————————————

program ALEX;
global
fuerza_salto=17;
vida=5;
municion=0;
sin_municion=TRUE;
enemigos=0;
fuente_peq;
s_morte;
s_bonus;
s_explos;
s_salto;
s_fondo;
s_intro;
canal1;
local
incx=0;
velocidad_gravedad=0;
en_suelo=FALSE;
begin
  set_mode(m640x480);

s_intro=load_pcm("pcm\enrique\intro.pcm",0);
  load_fpg ("fpg\enrique\alice\alex.fpg");
  put_screen(0,900);
  sound(s_intro,256,256);
  frame(15000);
  clear_screen();

fuente_peq=load_fnt("fnt\enrique\alicepeq.fnt");

s_morte=load_pcm("pcm\enrique\morte.pcm",0
);

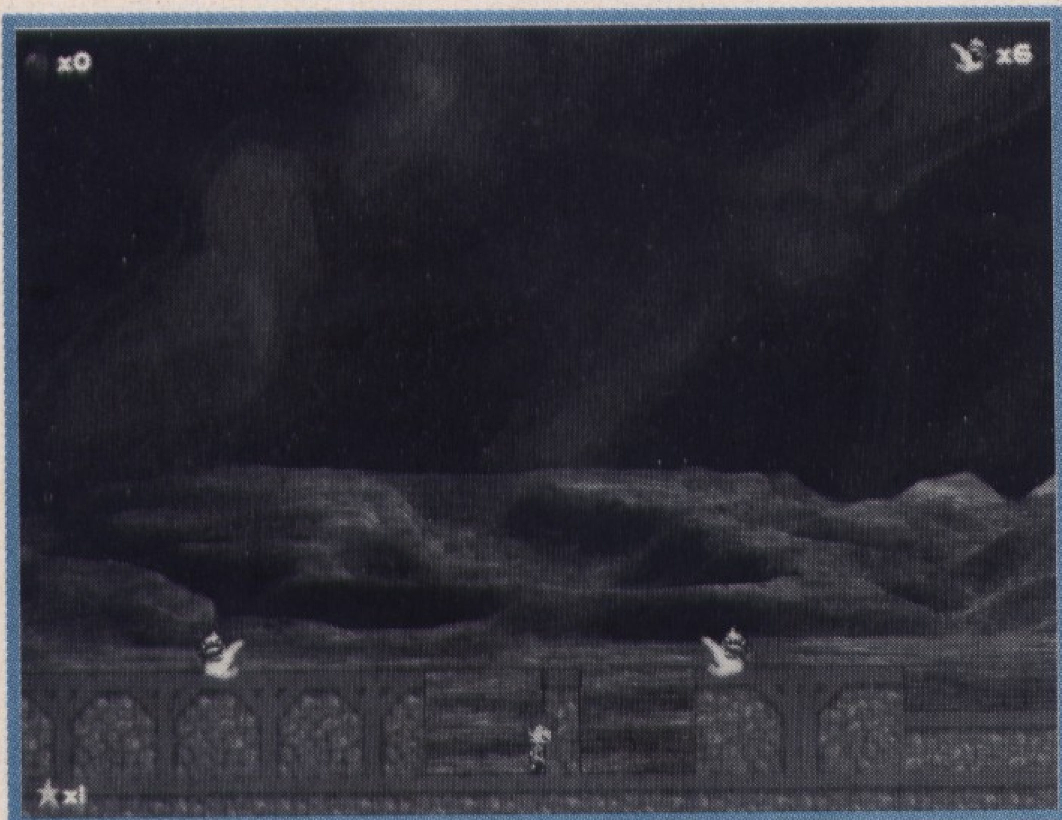
s_bonus=load_pcm("pcm\enrique\bonus.pcm",0
);

s_explos=load_pcm("pcm\enrique\explos.pcm",0
);

s_salto=load_pcm("pcm\enrique\salto.pcm",0);

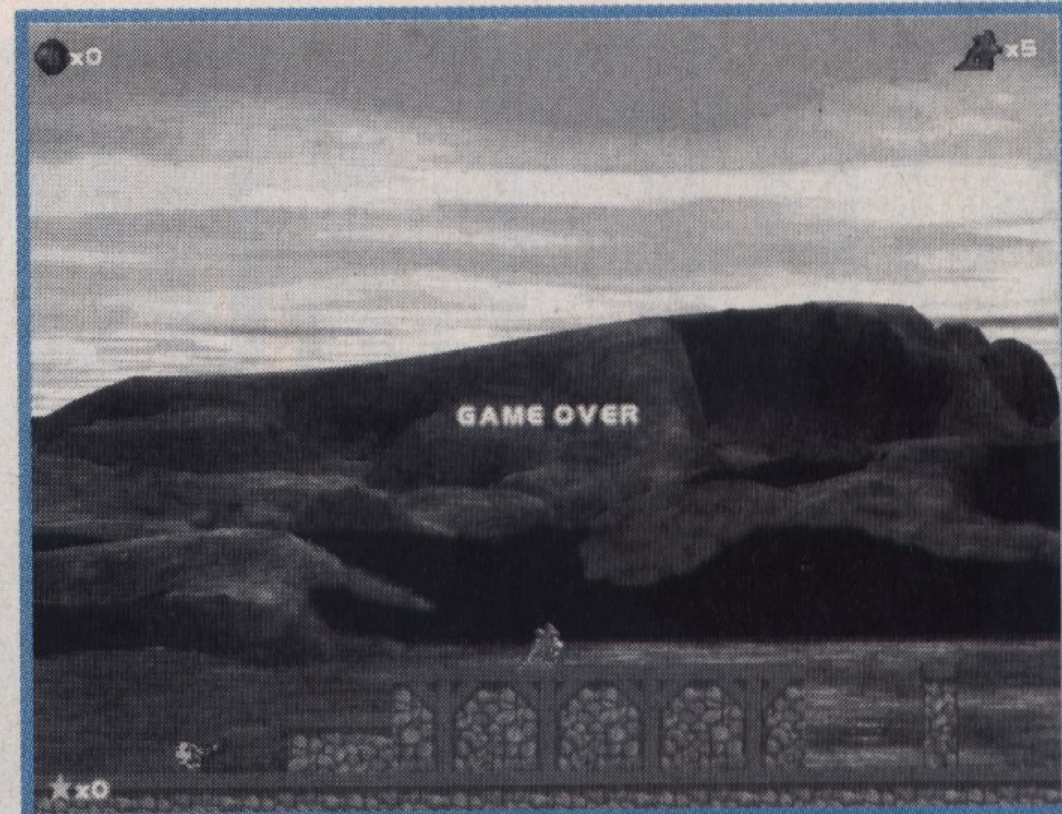
s_fondo=load_pcm("pcm\enrique\fondo.pcm",0)
;
  start_scroll(0,0,500,501,0,0);
  write (fuente_peq,25, 10, 0, "x");
  write_int (fuente_peq, 35, 10, 0, &municion);
  write (fuente_peq, 605, 10, 0, "x");
  write_int (fuente_peq, 615, 10, 0,
&enemigos);
  write (fuente_peq,25, 455, 0, "x");
  write_int(fuente_peq,35,455,0, offset vida);
  bomba();
  malo();
  chico_vida();
  personaje(70,395);
  enemigo (300,350,300,430);
  enemigo (665,350,665,740);
  enemigo (820,390,820,980);
  enemigo (1340,200,1340,1390);
  enemigo (1534,365,1534,1655);
  enemigo (1795,425,1795,1880);
  enemigo (1900,425,1900,2015);
  plataforma (515,415,415,445);
  bonus(515,250);
  vidas(1275,320);
  control();
end
process control();
private
id_txt;
begin
  ctype=c_scroll;
  timer=0;
  loop
  if(vida==0)
id_txt=write(fuente_peq,320,240,4,"GAME
OVER");
  frame(6000);
  stop_sound(canal1);
  delete_text(id_txt);
  exit("MARTOS GAMES, 2000",0);
  end
  if (enemigos==0) id_txt=write
(fuente_peq,320,240,4,"FIN DEL NIVEL 1");
  frame (10000);
  delete_text(id_txt);
  stop_sound(canal1);
  nivel2();
  end
```

Juegos ganadores: 3º

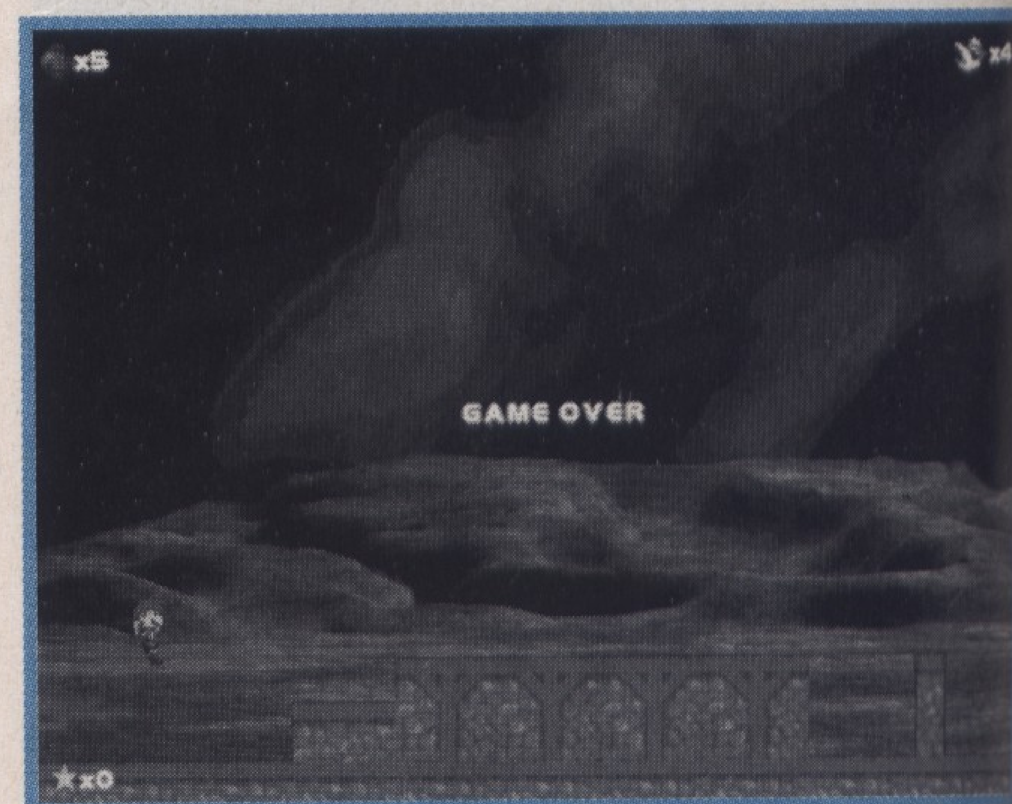


```
   if (municion==0 and not get_id(type bonus))
id_txt=write (fuente_peq,320,240,4,"—
ATENCION! Municion agotada");
   frame (10000);
   delete_text(id_txt);
   bonus (rand(20,2024),-40);
   end
   if (timer>100) timer=0; end
   if (key(_esc)) let_me_alone();break ; end
   frame;
   end
end
process control2();
private
id_txt;
begin
  ctype=c_scroll;
   timer=0;
   loop
   if(vida==0)
id_txt=write(fuente_peq,320,240,4,"GAME
OVER");
   frame(6000);
   stop_sound(canal1);
   delete_text(id_txt);
   exit("MARTOS GAMES, 2000",0);
   end
   if (enemigos==0) id_txt=write
(fuente_peq,320,240,4,"FIN DEL NIVEL 2");
   frame (10000);
   delete_text(id_txt);
   stop_sound(canal1);
   exit("MARTOS GAMES, 2000",0);
   end
   if (municion==0 and not get_id(type bonus))
 id_txt=write (fuente_peq,320,240,4,"—
ATENCION! Municion agotada");
   frame (10000);
   delete_text(id_txt);
   bonus (rand(20,2024),-40);
   end
   if (timer>100) timer=0; end
   if (key(_esc)) let_me_alone(); break; end
   frame;
   end
end
process  personaje(x,y);
private
```

```
pri_cuad=1;
ult_cuad=1;
disparo=1;
begin
  ctype=c_scroll;
  scroll.camera=id;
   canal1=sound(s_fondo,256,256);
   loop
      for  (graph=pri_cuad ; graph<=ult_cuad ;
graph=graph+1)
      if  (key(_left))
      incx=-10; pri_cuad=2; ult_cuad=9; flags=1;
      else if  (key(_right))
      incx=10; pri_cuad=2; ult_cuad=9; flags=0;
      else pri_cuad=1;  ult_cuad=1; incx=0; end
      end
      if  (key(_up) and en_suelo)
velocidad_gravedad=-fuerza_salto;
      sound(s_salto,256,256); end
      if (velocidad_gravedad<>0 or not en_suelo)
from graph=21 to 22; end end
      if (velocidad_gravedad > 0 and not
en_suelo) from graph=24 to 25; end end
      if (key(_space))
        if (disparo)
          disparo=0;
          if (municion>0) disparo_yo(x,y,flags);
municion—; end
        end
      else disparo=1; end
      gravedad(16,23);
      paredes (42);
      x+=incx;
      frame;
      end
   end
end
process muerto(x,y,direccion);
begin
  ctype=c_scroll;
  scroll.camera=id;
  vida=vida-1;
  stop_sound(canal1);
  sound(s_morte,256,256);
  graph=96;
  velocidad_gravedad=-10;
  if (direccion) incx=-10; else incx=10; end
  from angle=0 to 90000 step 6000;
  if (angle>40000) graph=97; end
  if (angle>60000) graph=98; end
  x+=incx;
  if (incx>0) incx—; else if (incx<0) incx++; end
end
  if (en_suelo) incx=0; angle=90000; graph=98;
end
  gravedad(11,11);
  frame;
  end
  loop
  x+=incx;
  if (incx>0) incx—; else if (incx<0) incx++; end
```




```
end
  if (en_suelo) incx=0; end
  gravedad(11,11);
  if (en_suelo) frame(3000); fade_off();
  personaje(70,350); fade_on(); signal (id,s_kill);
end
  frame;
end
end
process disparo_yo(x,y,direccion);
private
id2=0;
begin
  ctype=c_scroll;
  velocidad_gravedad=-20;
  size=50;
  if (direccion>0) incx=-20; else incx=20; end
  while (not en_suelo)
    from graph = 52 to 62;
    x+=incx;
    if (incx>0) incx—; else if (incx<0) incx++; end
end
    gravedad (11,11);
    paredes(23);
    frame;
    end
  end
  size=100;
  from graph = 63 to 75;
    id2=collision (type enemigo) or collision(type
enemigo2);
    if ( id2 )
     explosion (id2.x , id2.y, 50);
     sound(s_explos,256,256);
     signal (id2,s_kill);
```





El programa lector de Mapas de DIV en acción.

Leyendo el formato MAP

Para desarrollar un lector de cualquier tipo, en primer lugar, debemos tener en cuenta las capacidades del sistema operativo para el que vamos a programarlo. En nuestro caso trataremos con Windows 95. El dispositivo de la interfaz de gráficos de Windows o GDI, nos permite dibujar puntos de cualquier color con tan solo indicarle la posición donde queremos situarlo y los valores RGB del color, sin importarnos la resolución de la pantalla ni el número de colores. Si tenemos 256 colores, no podremos definir nuestra paleta adecuadamente, debiendo utilizar la propia de Windows. Sin embargo, Windows transforma automáticamente los valores RGB al color disponible más parecido, aunque esto puede distorsionar considerablemente la imagen. Si queremos tener control total sobre los controles de la paleta no tenemos más remedio que recurrir a las librerías DirectX, en especial a DirectDraw, que se encarga del manejo de la tarjeta de vídeo y sus capacidades 2D. Con esto, podemos disponer de aplicaciones en pantalla completa y cambiar a nuestro antojo la paleta de colores, pero como el grado de complejidad de esto último es mucho más alto, no vamos a entrar en detalle.

Una vez conocidas nuestras limitaciones, podemos pasar por tanto al proceso que debemos seguir para la lectura en nuestro sistema del archivo de mapa:

1. Para comenzar, debemos abrir el fichero del mapa como archivo binario de lectura y comprobar su correcta apertura.
2. En primer lugar, y como dijimos anteriormente, debemos comprobar el formato del archivo que vamos a leer, para no cometer posibles errores. Esto es muy importante, ya que por accidente podemos intentar leer un archivo de 65536*65536 puntos, y reservar memoria para ello, que no es poco. Por tanto, este debe ser nuestro primer paso.
3. Una vez comprobado que tenemos ante nosotros un archivo MAP, debemos reservar memoria para los datos de los puntos de la imagen. Para ello debemos considerar el ancho y alto de la pantalla, de forma que reservamos ancho x alto bytes para almacenar el mapa.
4. Acto seguido, debemos cargar la paleta en una tabla de 256 colores, cada uno de ellos con tres valores de color, rojo, verde y azul. Si bien se puede hacer más adelante, este puede ser un buen momento para multiplicar por 4 los índices de color para adaptarlos al sistema de Windows. Es recomendable realizar un desplazamiento a la izquierda de 2 bits sobre los valores en lugar de multiplicar por 4, ya que así se consumen menos ciclos de reloj y por tanto ganamos en velocidad.
5. La gama de colores no influirá en nuestro programa, por lo que debemos saltarnos el bloque dedicado a esta sección.
6. Debemos comprobar el número de puntos de control que tenemos en el mapa, ya que de esta forma podremos determinar la posición dentro del fichero del comienzo de la imagen. Esta posición vendrá determinada por la expresión:

Comienzo del mapa = 1394 + (4 x nº puntos de control)

Por tanto debemos colocar el puntero del fichero en dicha posición para así poder comenzar con la lectura de la imagen.

7. Ahora podemos ya proceder a la copia en la zona de memoria reservada anteriormente del mapa en sí.
8. Cerramos el archivo que no necesitaremos más.

Siguiendo este proceso hemos conseguido tener en memoria los datos que necesitaremos para procesar la imagen: la paleta y los puntos. ¿Qué debemos hacer ahora? Pintarla, ¿verdad?. Para ello, en primer lugar debemos tener algunos conocimientos básicos de la programación en Windows 95.

Conceptos básicos de programación

No es mi intención hacer un curso de Windows 95, sino introducir los conocimientos necesarios de la programación de esta aplicación para que pueda entenderse en su totalidad, y no crear ningún vacío en la comprensión de este pequeño programa. Es importante tener algunos conceptos de programación orientada a objetos o C++ para comprender algunos conceptos. Si no es así, no se preocupe en demasía, aunque hay algunas diferencias entre C y C++, no será un gran obstáculo. Sin más, procedamos a explicar algunos conceptos.

> Debemos comprobar el número de puntos de control que tenemos en el mapa

Por cuestiones didácticas, nuestro programa solamente pedirá el nombre del fichero a leer al comienzo y no podremos cargar ningún otro durante la ejecución. La razón de esto es la creación de una aplicación SDI, es decir, con una sola vista y documento abierto a la vez, más sencilla de desarrollar que una aplicación MDI o con varias vistas y documentos. Para pedir dicho fichero, utiliza un diálogo estándar llamado *CFileDialog*, que presenta el ya clásico menú de abrir fichero. Si pulsamos el botón de OK en dicho menú, la función devuelve un valor definido por IDOK, si no, recibimos un valor IDCANCEL. Para conocer el nombre del fichero al que hemos hecho referencia en el diálogo debemos consultar la función *CFileDialog::GetFileName()*, perteneciente a la clase *CFileDialog*. Este nombre se devolverá con el formato propio de la clase *CString*, que debemos convertir al formato están-



dar de *array* de caracteres para que la clase *fopen*, que es la que utilizaremos para abrir el archivo *map*, reconozca el nombre del fichero.

Para procesar convenientemente los datos del mapa a cargar, hemos creado una clase *CMapa*, que encapsula en comportamiento de la carga del fichero. En esta clase, está implementada una función llamada "CargarArchivo", que recibe como parámetro el nombre del fichero y almacena dentro de variables públicas (accesibles desde cualquier punto del programa) la paleta y la imagen.

Este proceso de carga del mapa, se realiza dentro de la sección "OnCreate" de la ventana del programa, es decir, esta secuencia de código se ejecuta durante la creación de la ventana, antes de ser mostrada. Una vez terminada la secuencia, la ventana se muestra, llamando a la función "OnPaint" para mostrar el contenido deseado en la vista. Esta función "OnPaint", se llamará automáticamente desde la propia ventana cuando sea necesario redibujar la imagen, bien porque hayamos cambiado de aplicación o bien alguien haya abierto un diálogo encima (acerca de, por ejemplo).

Podemos disponer de aplicaciones en pantalla completa y cambiar a nuestro antojo la paleta

El código de dibujo de la función "OnPaint", consta de un doble bucle que se encarga de recorrer horizontalmente la zona de dibujo, para de esta forma rellenar punto por punto la imagen. Para ello, en primer lugar obtenemos de la imagen el valor del color referido a la paleta. Posteriormente debemos situar el punto en la pantalla mediante la función "SetPixel", a la cual le debemos pasar las coordenadas de la pantalla y el valor RGB del punto que deseamos colocar. Esta es la secuencia que extraemos del programa:

```
SetPixel(dc,x,y,RGB(map->Paleta
[col][0]<<2,map->Paleta[col][1]
<<2,map->Paleta[col][2]<<2));
```

Aunque parezca complicado, en realidad sólo debemos tener en cuenta dos cosas:

1. El primer parámetro es un puntero, necesario para el dibujo en la pantalla, que recibimos en la propia función "OnPaint" como parámetro, de forma que no tenemos que preocuparnos por ello.
2. Los valores RGB, son el primer, segundo y tercer valor situados dentro del índice indicado por el color del punto en la paleta. A estos se les aplica un desplazamiento a la izquierda para multiplicarlos por 4 y así convertirlos al formato de Windows.

Si ahora ejecutamos el programa, observaremos que la carga de archivos de gran tamaño es más bien lenta. Esto se debe a que el dibujo se hace punto por punto. Si en lugar de cargar la imagen en un puntero, la procesáramos antes de dibujarla y la copiáramos dentro de una variable de tipo "Bitmap", podríamos posteriormente utilizar la función "BitBlt" para volcar directamente la imagen a la pantalla sin tener que calcular los valores y

Tabla 1. Estructura del formato MAP

CABECERA	
Identificador	"map" = 3 bytes
Código	1A0D0A00 (hexadecimal) = 4 bytes
Versión	0 = 1 byte
Ancho del mapa	1 word
Alto del mapa	1 word
Código del gráfico	1 double word
Descripción	32 bytes
PALETA	
Estructura de archivo PAL	1352 bytes
PUNTOS DE CONTROL	
Nº de puntos	1 word
(Por cada punto de control)	(Por cada punto de control)
Coordenada horizontal	1 word
Coordenada vertical	1 word
MAPA DEL GRÁFICO	
Puntos del mapa	(ancho x alto) bytes

Tabla 2. Estructura del formato PAL

CABECERA	
Identificador	"pal" = 3 bytes
Código	1A0D0A00 (hexadecimal) = 4 bytes
Versión	0 = 1 byte
PALETA	
256 colores	256*3 bytes
Por cada color	
Rojo (R)	1 byte
Verde(G)	1 byte
Azul(B)	1 byte
GAMAS DE COLORES (16 definiciones de gama)	
Nº de colores (8,16,32)	1 byte
Tipo de gama (0:directa, editable cada 1,2,4,8 colores)	1 byte
Fija (0 = no, 1 = sí)	1 byte
Colores de la gama	32 bytes máximo
MAPA DEL GRÁFICO	
Puntos del mapa	(ancho x alto) bytes

situarla punto a punto. Esto nos permite agilizar en gran medida toda la operación de dibujo. Es evidente que ésta sería una gran mejora, si bien, sería mucho más enrevesado como ejemplo para nuestro caso.

Pasemos a ver el código

En el cuadro 1 podemos observar el código de la clase "Cmapa", que se encarga de manipular los mapas de DIV. Es una clase sencilla y que ejecuta lo mínimo necesario para reproducir estos ficheros. Como al comienzo de este artículo dijimos, no es más que un posible esqueleto para futuras ampliaciones del programa. Con esto podemos hacer un conversor a formato PCX o cualquier otro y viceversa. El límite está en ti y tu imaginación. Espero que este artículo haya servido para despertar la curiosidad y el interés por experimentar en este basto campo aún poco explorado. Ánimo y feliz programación.

Para cualquier consulta o duda pueden mandar un e-mail a la dirección trinidad@arrakis.es

Pablo Trinidad



Código de la clase CMapa que decodifica los archivos MAP

Mapa.cpp: implementación de la clase CMapa.

```
#include "stdafx.h"
#include "MAPReader.h"
#include "Mapa.h"

#ifdef _DEBUG
#undef THIS_FILE
static char THIS_FILE[]=__FILE__;
#define new DEBUG_NEW
#endif
```

Constructor/Destructor

```
CMapa::Cmapa()
{
    imagen = NULL;
    b_cargado = false;
}

CMapa::~CMapa()
{
}
```

Función de carga de archivos MAP

```
bool
CMapa::CargarArchivo(CString
Archivo)
{
    FILE *file;
    char arch[200];

    // flag para indicar si se ha
cargado con exito
    b_cargado = false;

    // si había una imagen
anteriormente cargada, se libera
la memoria
    if (imagen != NULL) free
    (imagen);

    // conversión de tipos
necesaria para el uso del nombre
del archivo
    for (int i = 0; i <
Archivo.GetLength() ; i++)
        arch[i] = Archivo [i];
    arch[i]=0;

    if ((file = fopen(arch,"rb") ) ==
NULL)
    {
        MessageBox(NULL,"No
existe el archivo" +
Archivo,"Error",MB_OK);
        return false;
    }

    // leemos la cabecera y
comprobamos que se trata de un
archivo MAP
    fread(&cabecera,1,sizeof(Cabe
cera),file);

    if (cabecera.id[0]!='m' ||
cabecera.id[1]!='a' ||
cabecera.id[2]!='p'
        || cabecera.id[3]!= 0x1A ||
cabecera.id[4]!= 0x0d ||
cabecera.id[5] != 0x0A
        || cabecera.id[6]!= 0 ||
cabecera.id[7]!= 0)
    {
        MessageBox(NULL,"El
archivo no es de tipo
MAP","Error",MB_OK);
        return false;
    }

    // leemos los datos de la
paleta
    fread(Paleta,1,sizeof
(Paleta),file);

    // reservamos memoria para
la imagen
    if ( (imagen =
(BYTE*)malloc(sizeof(BYTE) *
cabecera.alto * cabecera.ancho))
== NULL)
    {
        MessageBox(NULL,"Error al
reservar
memoria","Error",MB_OK);
        return false;
    }

    // comprobamos el número de
puntos de control que tiene, y los
ignoramos
    fseek(file,1392,SEEK_SET);
    WORD temp;
    fread (&temp,1,2,file);
    fseek(file,1394 +
temp*4,SEEK_SET);

    // leemos la imagen
    fread (imagen,
1,cabecera.alto*cabecera.ancho,fil
e);

    fclose(file);

    b_cargado = true;
    return true;
}
```

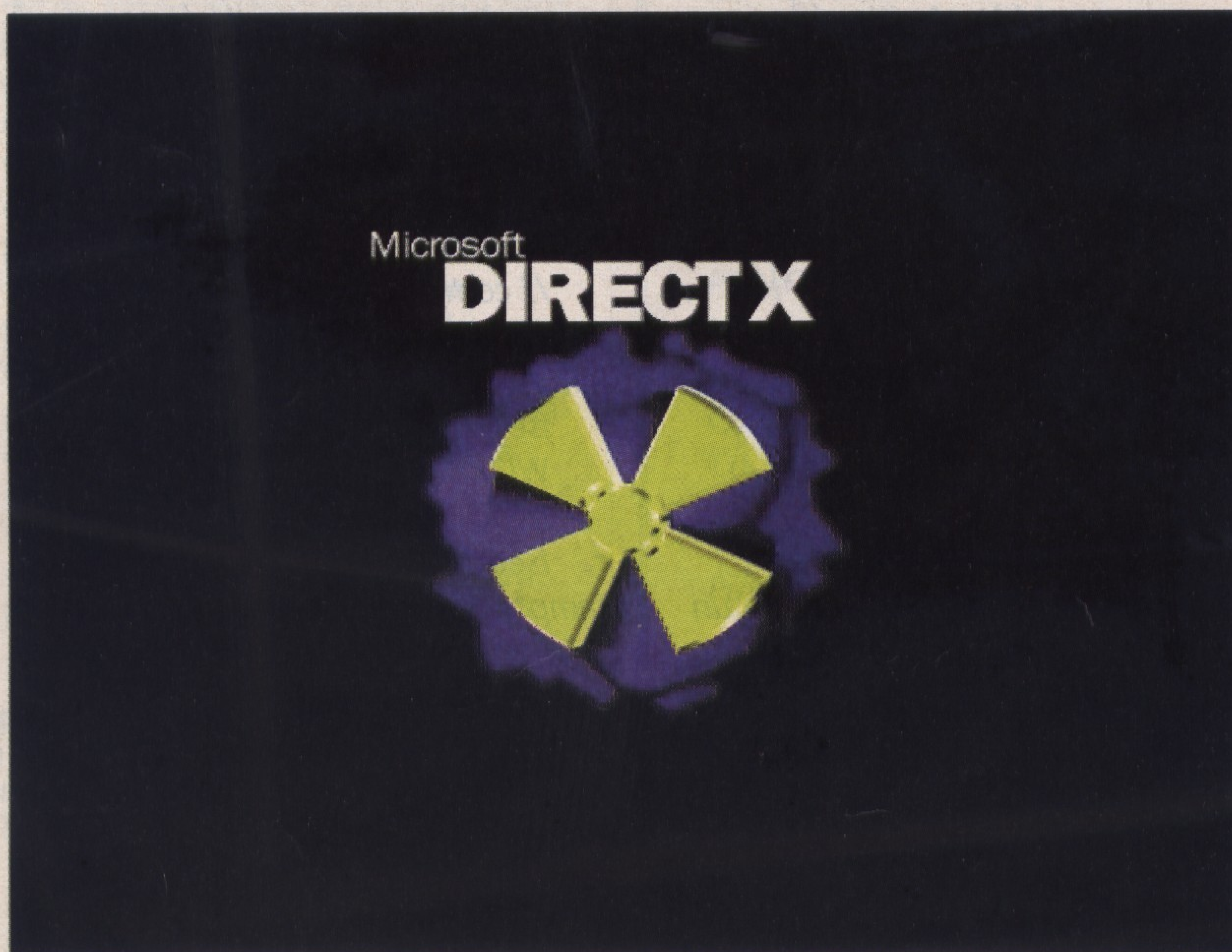
Libera la memoria asignada a la imagen

```
void CMapa::Libera()
{
    if (imagen != NULL)
    {
        free (imagen);
        imagen = NULL;
    }
}
```

Direct X

Conocer las superficies

La interfaz de DirectX que trabaja con los gráficos 2D es tal y como comentamos con anterioridad, la librería DirectDraw. Esta librería nos permite manipular con total libertad la memoria de vídeo, siendo además un paso necesario hacia cualquier aplicación que pretenda trabajar con DirectX.



Esto es lo que nuestro ejemplo debe mostrar, por supuesto podemos cambiar la imagen, respetando el tamaño.

Actualmente con las tarjetas gráficas podemos hacer algo más que volcar una imagen a la memoria de vídeo, el motor de las tarjetas, nos permite realizar distintos efectos con las imágenes, escalados, rotaciones, mezcla de imágenes, etc. Todas estas operaciones, podremos realizarlas fácilmente con DirectDraw, sin tener que preocuparnos si nuestra tarjeta soporta la rutina que queremos realizar. Si nuestra tarjeta no puede con ello, DirectDraw la emulará por software, con una merma lógica de velocidad, pero no podíamos esperar otra cosa.

Superficies

Las superficies son el alma de DirectDraw, son nuestra forma de acceder a la memoria de vídeo. Podemos crear superficies de cualquier tamaño, simplemente para guardar una imagen, o superficies del tamaño de la resolución de pantalla para volcarlas a la memoria de vídeo y mostrarlas.

Las superficies son básicamente contenedores de imágenes. Además de las que debemos crear para trabajar con la pantalla, casi siempre tendremos que crear algunas otras para almacenar los gráficos, una imagen o bien una secuencia de *sprites* que contengan una sencilla animación.

Cuando creamos un objeto DirectDraw, creamos al menos una superficie, que llamaremos superficie primaria, que es en realidad un acceso directo a la memoria de pantalla que vemos, es decir, todo lo que volquemos a la superficie primaria, es mostrada inmediatamente. Ahora bien, si mandamos directamente toda la información a la superficie primaria, cuando hagamos una animación comprobaremos un parpadeo, este parpadeo se debe al borrado del área en la que estaba nuestro *sprite*, y al volcado del nuevo *sprite*.

Evitar el parpadeo

Para evitar este desagradable efecto, crearemos una superficie auxiliar, esta segunda superficie nos servirá de intermediario entre la superficie primaria y las demás superficies que contengan nuestros gráficos. Es decir, la superficie auxiliar, será una copia virtual de la primaria en la que haremos las operaciones que necesitemos, una vez que hayamos completado nuestras operaciones, intercambiaremos la superficie auxiliar con la primaria, evitando de esta forma el molesto parpadeo de la animación.

Modo ventana, modo pantalla completa

Algo que debemos tener muy en cuenta en nuestro trabajo con las DirectX, es la diferencia sustancial que supone trabajar a pantalla completa o en modo ventana. Mientras que a pantalla completa somos los amos absolutos de la memoria de vídeo, en modo ventana, hemos de tener en cuenta que las demás ventanas tienen acceso a través del GDI a la memoria de vídeo. Varias ventanas a la vez pueden compartir la memoria de vídeo con nuestra aplicación, incluso pueden solaparse a la nuestra, por todo esto, debemos preparar las DirectDraw para tener en consideración esta contingencia.

Cuando utilizamos la configuración de pantalla completa,



Si tenemos instalado correctamente el SDK de DirectX, tendremos un nuevo icono en el panel de control.

DirectDraw nos provee de una superficie auxiliar unida a la primaria, además nos proporciona una función especifica que realiza el intercambio de forma totalmente trasparente para nosotros. Esta es una de las principales virtudes del modo pantalla completa. Ahora bien, si pretendemos trabajar en modo ventana, ya no disponemos de una superficie auxiliar, ni siquiera tenemos una función para realizar el intercambio, en su lugar, debemos crear la superficie auxiliar nosotros, y copiar el contenido de la superficie auxiliar a la primaria. El resultado es el mismo, pero con mucho más trabajo.

Modo ventana

Entre los problemas de trabajar en modo ventana no se encuentran tan solo el de no disponer de una superficie auxiliar, si no que además podemos encontrarnos con que alguna otra aplicación solapa nuestra ventana. Si no controlamos este hecho, podremos ver un desagradable efecto de intromisión de nuestra imagen con el área de otra ventana, algo que debemos evitar a toda costa. Para evitar este problema, contamos con un objeto de DirectDraw que nos permite resolver de forma automática este inconveniente.

Para iniciar el modo ventana debemos decirle a DirectDraw que queremos compartir la máquina con las demás aplicaciones, es decir que no queremos nada de prioridad ni tener en exclusiva ningún dispositivo ni área de memoria.

```
if (FAILED(lpDD->SetCooperativeLevel(hWnd, DDSCL_NORMAL))
    return false;
```

Modo pantalla completa

El modo ventana completa nos ofrece innegables ventajas, además de proporcionarnos la superficie auxiliar, contamos con la garantía de que ninguna otra aplicación tendrá la ocasión de interrumpirnos si nosotros no queremos, es más, podemos tomar prácticamente el control total de la máquina.

Iniciar el modo pantalla completa es sencillo, debemos decirle a DirectDraw que queremos el control total, y que además necesitamos toda la memoria de vídeo para mostrar nuestros datos a pantalla completa.

```
if (FAILED(lpDD->SetCooperativeLevel(hWnd, DDSCL_EXCLUSIVE | DDSCL_FULLSCREEN))
    return false;
```

Cargar una imagen

Una operación que realizaremos frecuentemente es crear una superficie para que contenga una imagen que tenemos en un archivo.

Como ejemplo sencillo, vamos a preparar las DirectDraw para trabajar en modo pantalla completa, prepararemos la superficie auxiliar, y crearemos otra superficie en la que cargaremos una imagen, mostrándola posteriormente.

En primer lugar, creamos la superficie primaria:

```
DDSURFACE ddsd;

ddsd.dwSize = sizeof(ddsd);
ddsd.dwFlags = DDSD_CAPS | DDSD_BACKBUFFERCOUNT;
ddsd.ddsCaps.dwCaps = DDSCAPS_PRIMARYSURFACE | DDSCAPS_FLIP |  DDSCAPS_COMPLEX;
ddsd.dwBackBufferCount = 1;

if (FAILED(lpDD->CreateSurface(&ddsd, &lpDDSPrimary, NULL))
    return false;
```

Con esta secuencia de instrucciones, hemos creado la superficie primaria. Como podemos ver, este proceso es bastante sencillo, tan solo debemos preocuparnos de la correcta creación de la superficie.

Una vez que tenemos la superficie primaria, obtendremos la superficie auxiliar. Al tener preparado DirectDraw para trabajar en modo pantalla completa, tenemos la superficie auxiliar a nuestra disposición por el sistema, sin que tengamos necesidad de crearla, tan sólo debemos reclamarla a DirectDraw.

```
ddscaps.dwCaps = DDSCAPS_BACKBUFFER;

if (FAILED(lpDDSPrimary->GetAttachedSurface(&ddscaps, &lpDDSBack)))
    return false;
```

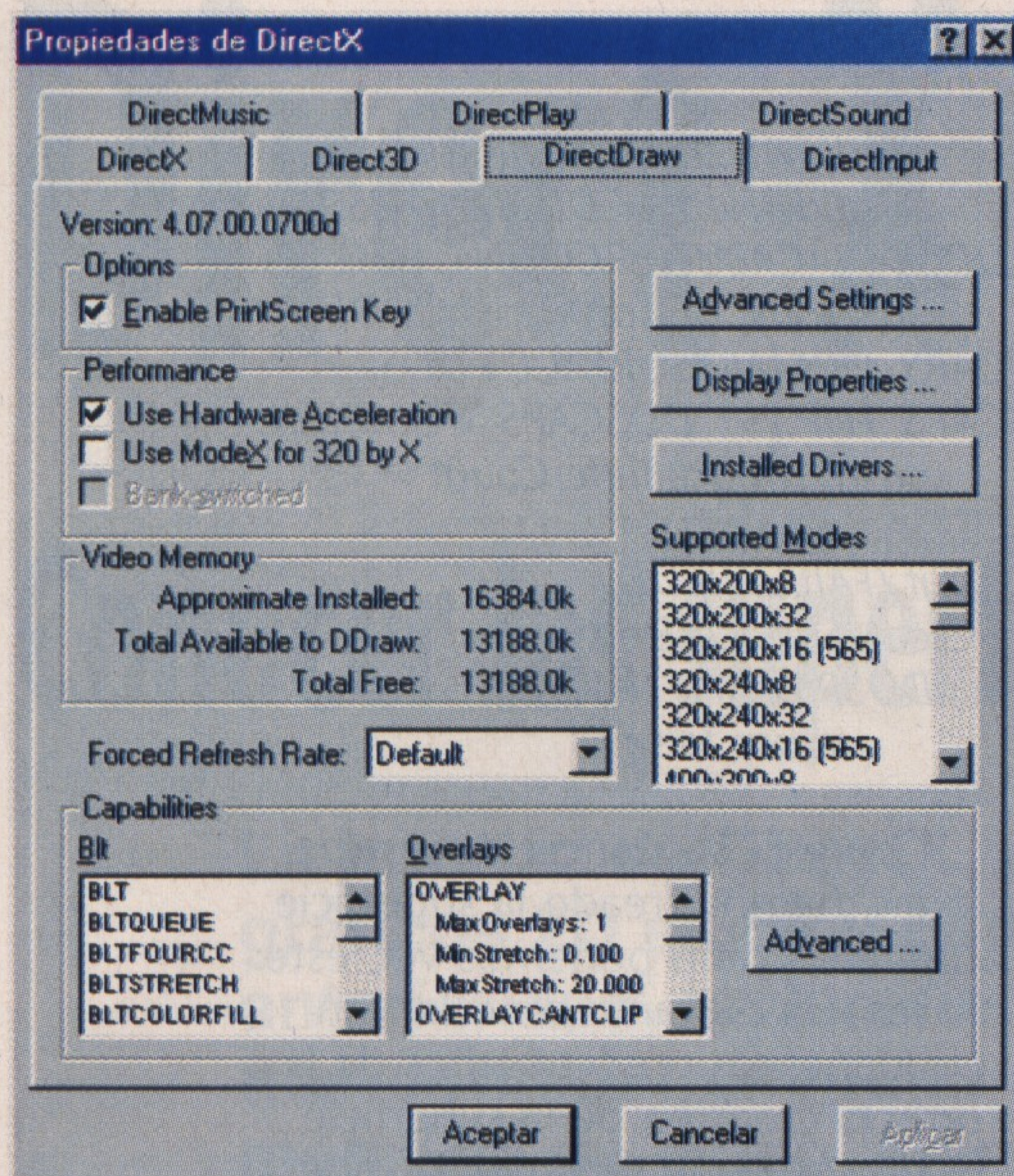
Gracias a esta superficie auxiliar evitaremos el parpadeo o la aparición de forma progresiva de la imagen.

> Debemos recordar liberar las superficies que hemos creado antes de cerrar nuestro programa

Para guardar una imagen, creamos una superficie del tamaño de la imagen que queremos cargar, en caso de que superficie e imagen difieran en tamaño, la imagen que contenga la superficie se verá deformada. Además, debemos indicar a DirectDraw que la superficie que queremos crear es de tipo no visible, esto es lógico, ya que su única misión será contener datos.

```
ddsd.dwSize = sizeof(ddsd);
ddsd.dwFlags = DDSD_CAPS | DDSD_HEIGHT | DDSD_WIDTH;
ddsd.ddsCaps.dwCaps = DDSCAPS_OFFSCREENPLAIN;
ddsd.dwHeight = 640;
ddsd.dwWidth = 480;

if (FAILED(lpDD->CreateSurface(&ddsd, &lpDDSurface, NULL)))
    return false;
```



Estas son las capacidades de nuestra tarjeta gráfica. DirectDraw nos proporciona acceso a ellas y emula las que no tengamos.

Una vez que tenemos la superficie creada, debemos cargarla y copiarla a la superficie. Este será un trabajo mixto entre las GDI de Windows, y DirectDraw.

El correcto uso de las superficies es necesario para aprovechar la potencia de DirectDraw

```
HBITMAP hbm = (HBITMAP) LoadImage(NULL,
"imagen.bmp", IMAGE_BITMAP,
640, 480, LR_LOADFROMFILE);

if (!hbm)
    return false;
```

Con estas instrucciones, si hemos tenido éxito, habremos cargado la imagen en un objeto creado a tal efecto.

A continuación debemos copiarla en nuestra superficie no visible. Podríamos cargarla también en la superficie auxiliar directamente, pero si queremos conservarla es mejor cargarla en una superficie creada a tal efecto.

```
HDC hdcImage;
HDC hdcSurf;

hdcImage =
CreateCompatibleDC(NULL);

SelectObject(hdcImage, hbm);

if (lpDDSurface->GetDC(&hdcSurf))
    return false;

if (!BitBlt(hdcSurf, 0, 0, 640,
480, hdcImage, 0, 0, SRCCOPY))
    return false;
```

Ya tenemos copiada la imagen en la superficie, así que el resto de los datos podemos liberarlos.

```
if (hdcSurf)
    lpDDSurface-
>ReleaseDC(hdcSurf);

if (hdcImage)
    DeleteDC(hdcImage);

if (hbm)
    DeleteObject(hbm);
```

Ahora, tan solo nos resta mostrar la imagen que hemos cargado, y para eso, copiaremos el contenido de la superficie no visible, a la superficie auxiliar, intercambiando posteriormente su contenido con la superficie principal.

```
lpDDSBack->BltFast(0, 0,
lpDDSurface, NULL, 0);

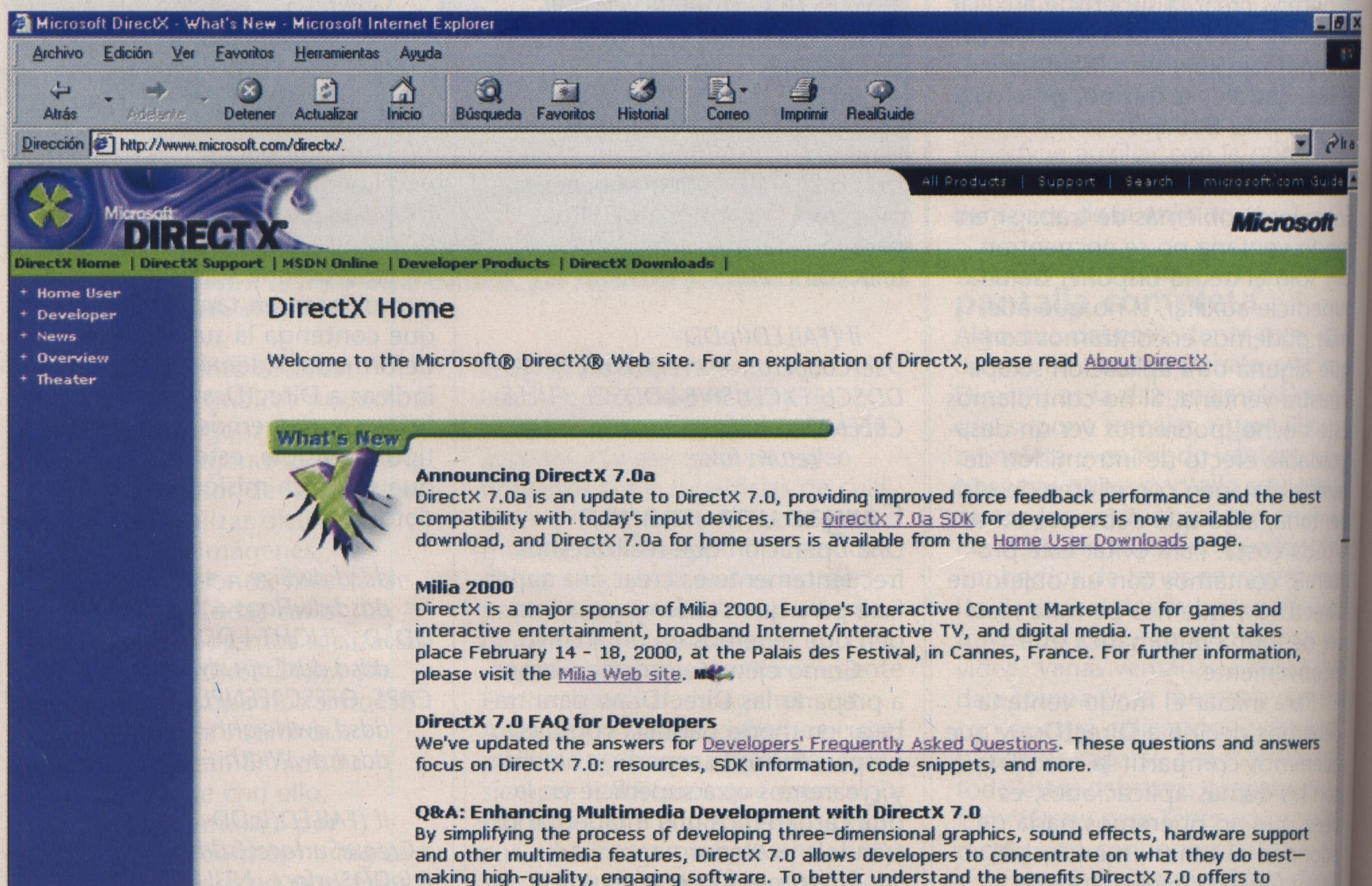
lpDDSPrimary->Flip(NULL,
DDFLIP_WAIT );
```

Con esta simple orden hemos intercambiado, de forma totalmente trasparente para nosotros, el contenido de la superficie auxiliar y la principal. Al mismo tiempo, le hemos indicado a DirectDraw que debe esperar a que la operación haya concluido para devolvernos el control.

Para finalizar, debemos recordar liberar las superficies que hemos creado antes de cerrar nuestro programa, hay que ser educados con el sistema y evitar dejar en memoria basura que irá comiendo espacio en nuestra exigua memoria.

En cualquier caso, las dudas que tengáis podéis remitirlas a la siguiente dirección de e-mail: gover@prensatecnica.com, por mi parte, intentaré resolver aquellas que mis limitados conocimientos me permitan.

Armando Vélez



Esta es nuestra dirección de referencia en Internet : www.microsoft.com/directx.

Curso de juegos en 3D

DIV Deathmaker (II)

En este número veremos lo importante que es tener planeado y diseñado el juego antes de comenzar a hacerlo. Una buena planificación y un buen diseño son elementos primordiales a la hora de ponerse manos a la obra para realizar la ardua tarea de programar un buen juego.

Normalmente, nadie dice "quiero hacer un juego" y luego mientras lo hace se le va ocurriendo cómo va a ser, qué tipo de juego es, etc. Por regla general, en los juegos que hagamos en 3D (como en todos) debemos tener bien diseñado el juego antes de empezar, puesto que si no luego nos podremos perder en la programación, o el juego puede quedar un poco "soso", o querremos introducir nuevas cosas en el juego sobre la marcha, obligándonos a retroceder en lo que llevábamos hecho constantemente

Nuestro *Div Deathmaker* está planeado por encima, sin embargo todavía quedan bastantes aspectos por determinar.

Los Bots

¿Cómo van a ser nuestros BOTs? Con el juego no va a venir ninguno ya creado, por lo que tendremos que editar los nuestros. Para ello, cada BOT tendrá una serie de parámetros que se irán guardando en una estructura STRUCT bot[29].



Adaptar los gráficos a una paleta fija no es tarea fácil.



Este singular personaje forma el skin base de nuestro juego.

¿Y por qué va a tener 30 registros nuestra estructura (contando con el nº0)? Porque podremos crear hasta 30 bots distintos, almacenándolos todos en un archivo. Se podría hacer de tal forma que no hubiese límite de bots (sólo el límite del disco duro) pero sería complicarlo mucho, y para un juego como éste, cuyo fin es didáctico, no hace falta complicarse tanto.

En nuestro menú habrá una opción *BOTS* que permitirá editar y borrar los BOTs. Dentro de cada registro (o sea, cada BOT) habrá una serie de datos:

NOMBRE: por supuesto, todos los BOTs tienen que tener un nombre. Nota: hay que ser originales, que lo de poner el nombre de un profesor o del jefe está muy visto ya. ;) Este dato se almacena en una cadena de texto.

SKIN: será una cadena de texto también, que contendrá la ruta y el nombre del FPG que usaremos como skin para este bot. Por ejemplo,

"c:\div2\dd\skin\exnovia.fpg".

AGRESIVIDAD: de 0 a 100, cuanto mayor sea más agresivo será el BOT. Esto es, si tiene 100, al menor indicio de vida comenzará a disparar como loco (comúnmente llamada berserker), si tiene un lanzamisiles no dudará en disparar aún teniendo al enemigo a un palmo, y será muy persistente, es decir, que si ve a alguien no parará hasta matarle. Si es 0 será mucho más precavido, y apuntará primero y disparará después, y si se ve malherido intentará alejarse del combate.

PUNTERÍA: eso mismo; también variable entre 0 y 100, cuanto mayor sea mejor apuntará, y cuanto menor sea, peor.

AGILIDAD: determina la velocidad a la que se mueve el BOT y la velocidad de giro, rapidez con que apunta, etcétera. También varía de 0 a 100.

PERCEPCIÓN: de 0 a 100 determina cuándo debe darse cuenta un BOT de que hay otro cerca y cuando no. Si es baja, no verá los enemigos que estén lejos y los que tenga cerca a los lados, posiblemente tampoco. Si es alta, aún si estás detrás es posible que te detecte.

ARMA PREFERIDA: de 0 a 7, especifica si tiene algún arma preferida y cuál es. Si vale 0, no tendrá arma preferida, es decir, disparará con la última arma que haya recogido o con la siguiente que tenga si se le acaba la munición de una. Si es de 1 a 7, será:

- Láser de impulsos.
- Qk22.
- ADC (Ametralladora de Combate).
- Pistola Bláster.
- Repetidor láser.
- Lanzamisiles.
- Mina antipersonal.

Si tiene alguna preferida, siempre que tenga ese arma con munición cambiará la que tenga por esa, y la procurará usar siempre para atacar.

Si miráis la declaración de variables globales en el archivo DD.PRG encontraréis la declaración de esta estructura:

```
STRUCT bot[29];     //estructura que contiene parámetros de los bots
    string nombre;  //nombre del bot
```



Este aspecto tenía la ADC antes de cambiarle la paleta.

string skin; //ruta y nombre del FPG (skin)
int agr; //agresividad
byte atrib[3]; ////puntería[0],agilidad[1],percepción[2],arma preferida[3]

END

Os extrañaréis al encontrar que *AGR* es un dato de tipo *INT* cuando, al poder oscilar sólo entre 0 y 100, sería más conveniente declararlo como byte (de 0 a 255, sólo ocupa 1 byte de memoria y no 4) dentro de la tabla *atrib*. Pero en realidad no es más conveniente, sino todo lo contrario. Si suponemos que *agr* es un byte y sumamos nos quedan 5 bytes, con lo que *atrib[4]* ocuparía 8 bytes de memoria, quedando tres libres. Hemos obviado las *STRING* ya que tienen un valor divisible por 4. Según esto, quedan 3 bytes desaprovechados por cada registro de *bot[29]*, es decir, ¡30·3=90 bytes de memoria desperdiciados! Sin embargo, si ponemos *AGR* como dato de tipo *INT*, no queda ninguno libre, ya que los 4 bytes, al estar en una tabla, ocupan una alineación de memoria entera (4 bytes), el *int* otra (otros 4) y los *string* lo que ocupen (son 256 bytes, por dos, 512). Total, que ahora ocupa lo mismo que del otro modo, pero los 90 bytes no están desperdiciados, sino que están haciendo que se acceda a *AGR* mucho más rápido (ya que los datos de tipo *INT* se gestionan más rápidamente que los de tipo *BYTE*), cosa que nos viene de perlas ya que constantemente estaremos accediendo a esa variable.

Puedes configurar los bots dándoles la agresividad y habilidad que tu quieras

Las armas

Ya están todas diseñadas y preparadas para ser incluidas en el juego; las podéis encontrar en el FPG *"WEAPONS.FPG"*. Es verdad que ocupa mucho el *FPG*, pero esto es sólo porque todos los gráficos de las armas están a 640x400 (aunque las armas sólo ocupan en el gráfico su posición relativa donde aparecerá luego en el juego), para poder manejarlos fácilmente luego, ya que tienen el punto de control en 320x200, justo el centro de la pantalla. Si ocupan tanto, es porque los *MAP* (que es el tipo de gráfico de los *FPG*) son imágenes tipo *BITMAP* (como el *BMP*) no comprimidas, que son después de todo una matriz de 640 por 400 donde indica el color de cada pixel. Pero no hay problema, ya que una vez compilemos y hagamos el ejecutable, DIV se encargará de comprimir estos archivos en un *PAK*, y el *WEAPONS.FPG* pasará de ocupar 3.5 Mb a ocupar... ¡¡67 K!! Sí, sí, como leéis, sólo 67 K.

Hay dos imágenes por arma, una en la que el arma está quieta, y en la otra disparando. No he complicado mucho las cosas por razones obvias; esto no es un juego profesional sino un juego-ejemplo para un curso.

El proceso que controla todo

Bueno, he aquí una técnica que yo uso siempre en mis juegos y da buen resultado, y pienso que todo buen engine debería usarla.

Nuestro proceso principal, el *PROGRAM Div_Deathmaker*, va a ser el que controle en todo momento cómo están las cosas y qué debe hacer, a que procesos llamar, etcétera. Es un proceso que estará siempre activo comprobando el estado.

¿Y qué es el ESTADO? "Estado" es una variable global que tiene una "posición" asignada a cada valor posible. Esa "posición" debemos asignarla nosotros. No estaría mal apuntarnos en un papel "Estado=1 → está en el menú principal ; Estado=21 → está en los créditos tras haber muerto en la fase 57"... esto es sólo un ejemplo. Luego existe el proceso principal (que no tiene que ser el propio PROGRAM, yo suelo usar un proceso específico llamado "encargado") que controla los valores de esa variable de este modo:



El lanzamisiles hará las delicias de los más agresivos.

LOOP
IF (estado= =un_valor)
llama a los procesos necesarios;
haz lo que tengas que hacer para prepararlo;
WHILE (estado= =ese_valor)
loqueseanecesario() ;
FRAME ;
END
mata a los procesos que haga falta ;
descarga lo que sea necesario ;
END

// y luego igual con otro valor, y otro, y otro.

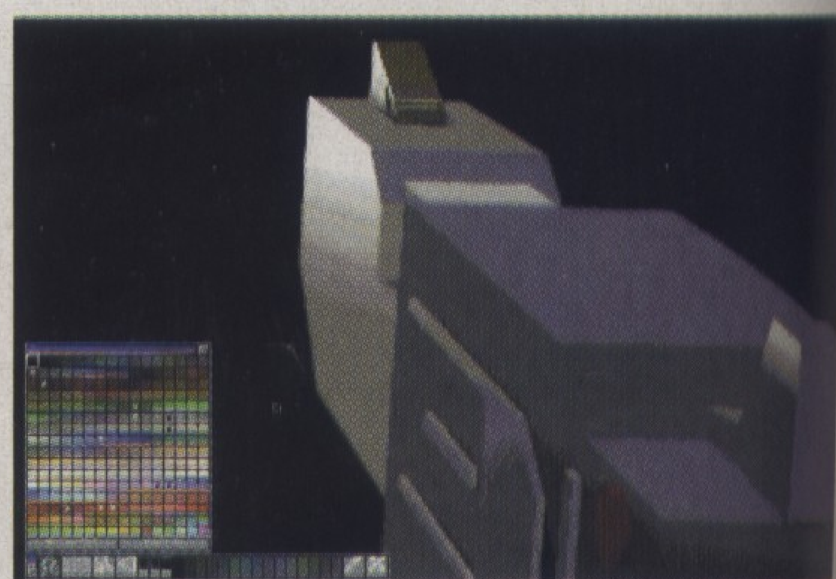
FRAME;
END

Es una técnica buena y útil, sobre todo para manejarnos por los menús, siempre y cuando no sean menús idénticos todos, y no sea un juego muy "monótono". En *Div Deathmaker* se usa esta técnica.

En el CD

Dentro del CD de DIVmania podréis encontrar, en el directorio de 3D/Red, los siguientes archivos pertenecientes a nuestro juego *Div Deathmaker:*

- **TITLE.PCX:** es la portada del juego; viene aparte en un PCX porque contiene una paleta distinta a la estándar del juego. Es la pantalla que aparece nada más iniciar el juego.
- **MENU.FNT:** es el tipo de letra que usamos para el menú, para las opciones a elegir, los textos grandes, etc.
- **MENSAJE.FNT:** es el tipo de letra estándar pequeño de nuestro juego, usada para mostrar los mensajes durante el juego, y algunos textos de los menús.
- **DEFAULT.FPG:** es el SKIN base de nuestro juego. Un *skin* es un conjunto de imágenes y/o sonidos de un programa, que pueden ser intercambiados fácilmente por otros hechos por un usuario. Este *skin* es el único que incluimos; para evitar que todos los bots tengan el mismo aspecto alguien tendrá que currarse más skins ^_^. Como podéis ver es



Siempre hay que retocar un poco los gráficos tras adaptarlos a la paleta.



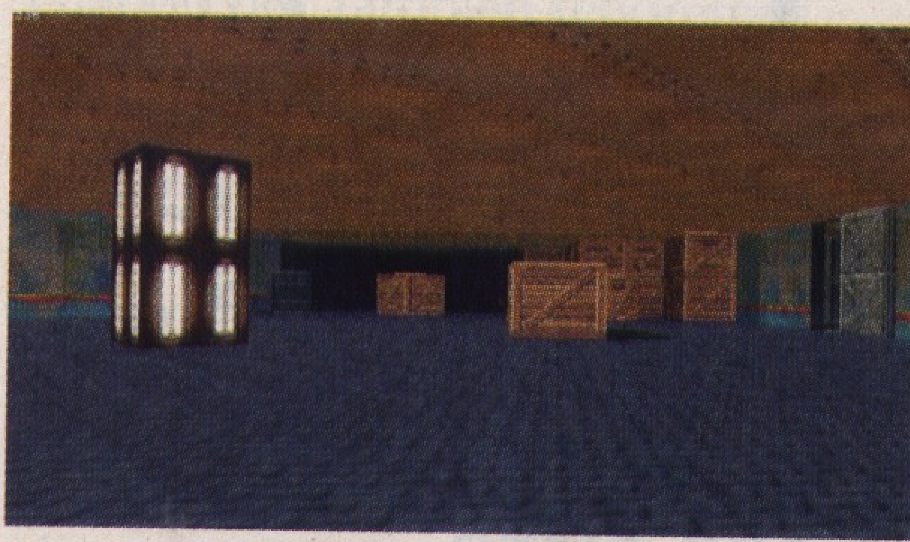
Cuadro 1. FPG para un SKIN

1-5:	personaje corriendo, ángulo 0º
6-10:	personaje corriendo, ángulo 45º
11-15:	personaje corriendo, ángulo 90º
36-40:	personaje corriendo, ángulo 315º
41-43:	1 personaje quieto + 2 personaje disparando, ángulo 0º
44-46:	1 personaje quieto + 2 personaje disparando, ángulo 45º
47-49:	1 personaje quieto + 2 personaje disparando, ángulo 90º
62-64:	1 personaje quieto + 2 personaje disparando, ángulo 315º
65-79:	personaje muriendo, ángulo 270º (mirando hacia la cámara)

una especie de androide con gafas de sol y máscara filtradora.

- **TEXTURES.FPG:** es el mismo WLD_VIEW.FPG que viene con DIV 2, adaptado a nuestra paleta estándar del juego. Serán las texturas que usemos para nuestros mapas.
- **DD.PAL:** es la paleta estándar; durante el juego (a partir de que aparece el título) el juego forzará la paleta como la única disponible, es decir, todos los gráficos que carguemos se adaptarán a esta paleta de colores.
- **WEAPONS.FPG:** son las armas, es decir, los dibujos de las armas que aparecerán en nuestra "mano" mientras juguemos; están todas diseñadas, desde el débil láser de impulsos hasta el potente lanzamisiles. Os aseguro que su aspecto recién renderizadas era magnífico, pero al limitar la paleta a 256 colores y compartirla con las paletas de las demás armas, texturas, etc., han perdido bastante calidad.
- **MISC.FPG:** se llama así por "miscelánea", contendrá cosas como el punto de mira, los iconos de vida y armadura, los objetos a recoger, etcétera. Pero de momento, está vacío.
- **DD.PRG:** es nuestro programa, aunque no tenga mucho por dentro todavía.
- **WLDVIEW2.WLD:** es el mapa *WLD_VIEW.FPG* que viene con DIV 2, adaptado como si fuera un mapa de *Div Deathmaker*. Contiene todas las banderas, tanto las opcionales como las obligatorias.
- **CUSTOM.TXT:** es un archivo TXT que me gustaría que incluyerais con cualquier *skin* o mapa que diseñéis para *Div Deathmaker*. Sólo tenéis que rellenarlo y cambiar el nombre de CUSTOM por el nombre del archivo del *skin* o mapa. Por favor si alguien diseña alguno, que me lo mande a ferminho@lycosmail.com

Como podéis ver, el PRG no está más que empezado, con muy pocas líneas, dado que es mejor terminar de diseñar por completo el juego antes que comenzar a programar. Para que funcione el PRG, y mejor que lo hagáis, cread un directorio DD\ dentro de vuestro directorio de DIV 2 (por ejemplo, C:\DIV2\DD) y meted ahí todo.



Aprovechando que ya está hecho, podremos usar WLDVIEW2 para jugar.

Haciendo skins

Para crear nuestros propios skins no hace falta más que... Div. Tenemos que crear un *FPG* con los gráficos del cuadro 1.

Hay que tener en cuenta que luego el juego adaptará el *FPG* del *skin* a la paleta estándar, por lo que mejor será hacer los gráficos de este *FPG* directamente en esa paleta (para evitar una pérdida muy grande de calidad al convertir la paleta). Para que encajen perfectamente los gráficos, éstos deben tener el punto 0 de control justo en el centro horizontal del gráfico, a la altura de los pies del personaje. Los tamaños pueden variar dependiendo del propio tamaño del personaje que se le quiera dar, pero el estándar ronda los 120-130 pixeles de alto y 80-90 de ancho.

Haciendo escenarios

Un factor a tener en cuenta es que al jugar, *Div Deathmaker* pide el nombre del *WLD*, es decir, del mapa 3D que vamos a usar. Aunque viene un mapa con *Div Deathmaker*, hecho por mí (al igual que el *skin*), al final termina aburriendo y uno se lo conoce demasiado bien. Por eso, vamos a ponerlo fácil para que cualquiera que tenga DIV 2 pueda diseñar un mapa para *Div Deathmaker* sin muchas complicaciones.

El mapa que pienso incluir no está todavía acabado, pero el método para hacer fácilmente mapas ya lo he pensado: solamente hay que diseñarlo con el *FPG* de *TEXTURES.FPG* que viene con *Div Deathmaker*, teniendo en cuenta que ciertas banderas tienen ciertos usos, que vienen detallados en la tabla. El archivo *WLDVIEW2.FPG* es un buen ejemplo, vienen situadas todas las banderas, aunque por supuesto se pueden omitir las que se quieran, salvo una del lugar de comienzo, que es necesaria y obligatoria. Todas las demás son prescindibles.

Bueno, así parece todo bastante fácil ¿verdad? Para el próximo número tendremos el mapa estándar (hasta entonces id mirando el *WLDVIEW2.FPG* si tenéis dudas), cómo configurar los BOTs salvando y cargando los datos, y empezando la partida. Recordad que si tenéis alguna sugerencia/duda/o lo que sea me lo podéis mandar a mi e-mail, las 24 horas del día los 365 días del año. ¡Hasta el próximo número!

En el escenario donde se desarrolla la lucha deben ser colocados los items oportunos

Ferminho (Fermín Vicente)
ferminho@lycosmail.com

Cuadro 2. BANDERAS en la edición de mapas para Div Deathmaker

1-10:	lugares de comienzo. Con 1 ya vale, pero sería una auténtica cosechadora de la muerte (muerte por desintegración).
11:	ADC (Ametralladora de Combate).
12:	Bláster de fusión.
13:	Repetidor láser.
14:	Lanzamisiles.
15-24:	Minas (para coger).
25-29:	ReGens pequeños.
30-32:	ReGens medianos.
33-34:	ReGens grandes.
35-39:	Blindajes pequeños.
40-42:	Blindajes medianos.
43-44:	Blindajes grandes.
45-49:	Cargadores de munición.
50:	Caja de cargadores.
51-59:	Células compactas.
60-61:	Baterías.
62-63:	Cajas de misiles.
64:	Quad Damage.
65-80:	Cajas explosivas.

Del monólogo al diálogo

Conversando con los personajes

Hasta ahora hemos conseguido que nuestro personaje mantenga un monólogo que nos permite saber, por ejemplo, qué mira o qué opina de lo que sucede a su alrededor. Ahora es tiempo de que dialogue con otros personajes, parte vital en toda aventura gráfica que se precie.



Es muy importante en todo juego de aventuras el comunicarse. Cuando el personaje mira un objeto, cuando intenta hacer una acción que no puede realizar, e incluso cuando opina algo importante que el jugador debe saber, se lo dice en una especie de monólogo, con frases como: no veo nada en especial, es demasiado pesado para moverlo, no creo que eso funcione... Esto ya lo hemos resuelto. En el artículo anterior comprobamos la importancia que hubiese otros personajes que ayudasen o entorpeciesen al protagonista de nuestra aventura, siendo muy interesante el poder hablar con ellos. Ya desde el principio, cuando definimos nuestra primera aventura, contemplamos la acción "hablar".



El diálogo entre un personaje secundario y el protagonista puede convertirse en algo tan simple como queramos o tan complejo como podamos. Usar PSI (personajes pseudo inteligentes) nos permitirá dar mayor atractivo a la aventura, aunque el reto es mayor. Conseguir que el diálogo varíe según lo que nuestro personaje vaya realizando da un mayor realismo. Eliminar las respuestas posibles de un diálogo cuando ya se han usado o bien cuando ya no tiene sentido usarlas, es una recomendable medida para evitar la "monotonía" en el juego.

Para empezar a preparar todo lo necesario para los diálogos en nuestro programa seguiremos con nuestro código de prueba (del número 4 de la revista). El programa hasta ahora nos permitía usar un arco con una flecha para tener un arco cargado, usar una diana con un trípode para tener una diana segura y, por último, usar el arco cargado con la diana para darle con la flecha justo en el centro. En el proceso diana, hemos recuperado parte del código fuente anterior (del número 3 de la revista) que hacía que el nombre y descripción de la diana fuese variando cuanto más investigásemos sobre ella, es decir, cuanto más veces la mirásemos. Hasta ahora siempre hemos hecho que lo que quiera que deba suceder cuando el protagonista actúe sobre un objeto, esté programado en ese objeto. De la misma forma, lo que quiera que el protagonista pueda decir o deba escuchar de un personaje, lo programaremos directamente en ese personaje.

Planeando

Necesitamos que cuando el jugador indique al protagonista hablar con algo, si se puede hablar con





ese objeto, presente una serie de opciones de diálogo. Para empezar, está muy bien tener de una a cuatro opciones de diálogo (cuatro frases alternativas), de las que una siempre será abandonar la conversación. Una vez que se escoja una de las opciones, se debe decir en voz alta y el objeto con el que hablamos debe responder. Tras la respuesta, la conversación finaliza o continúa. Vamos a usar el objeto "diana". Haremos que el protagonista, según lo que sepa de la diana hasta ese momento, pueda decirle unas cosas u otras. Según las veces que la miremos, nuestro protagonista la verá como: cosa, posible diana, dudosa diana, segura diana o diana de ejemplo. Las frases que el protagonista puede usar y sus correspondientes respuestas serán:

Protagonista (P); Diana (D):
P: Hola, ¿eres una nombre?
D: Puede que sí, puede que no.
P: Pareces una nombre.
D: Hasta puede que sea nombre.
P: No me gustan las nombre.
D: Todas las nombre no se comen.
P: Adiós.
D: Ciao.

Los textos cambiarán según la situación, sustituyendo nombre por el nombre del objeto en ese momento. Por ejemplo, la primera vez será "hola, ¿eres una cosa?". Y la siguiente, después de haberla mirado, "hola, ¿eres una posible diana?". La última de las frases



(adiós), nos despedirá de la diana y finalizará la conversación.

Preparando lo nuevo

Primero hemos de hacer algunos cambios en nuestro código actual. Las opciones del diálogo se presentarán en la parte inferior de la pantalla. Para esto, debemos hacer desaparecer el menú de acciones y la bolsa de objetos. Creamos un "flag" llamado *menu* que nos indicará cuándo está el menú activo y cuándo no. También tendremos *amenu*, que recordará el estado anterior de *menu*. Cuando *menu* sea cierto (menú activo) se ejecutará el bloque del control del menú de opciones; si no, mientras el menú esté inactivo, al pasar el ratón por la parte inferior de la pantalla no sucederá absolutamente nada. Cuando detectemos que *menu* y su estado anterior *(amenu)* no sean iguales, estaremos seguros de que su valor ha cambiado, procurando hacer:

- Si *menu* es falso (menú inactivo). Antes estaba activo, debemos ocultar el menú, quitar los botones de bolsa de objetos, ocultar los objetos contenidos en la bolsa, deseleccionar cualquier acción posible.
- Si *menu* es cierto (menú activo). Antes estaba inactivo, debemos mostrar el menú, poner los botones de bolsa de objetos, mostrar los objetos contenidos en la bolsa.

Con este fin creamos los siguientes procesos:

- *activa_menu() y desactiva_menu()*. Se limitan a asignar verdadero o falso a la variable global *menu*. Crear funciones para este proceso es recomendable para conseguir mayor legibilidad en el código. Es más fácil entender qué hace *desactiva_menu()* que *menu=FALSE*.
- *bolsa_inicia()*. La modificamos para que sólo se encargue de iniciar las variables.



- *bolsa_muestra()*. Crea los botones de control de la bolsa (arriba y abajo) y la dibuja (antes realizado por *bolsa_inicia*)
- *bolsa_oculta()*. Sitúa todos los objetos de la bolsa fuera de la pantalla y mata los procesos de los botones de control de la bolsa.
- *pon_menu()*. Pone la imagen de fondo del menú. Este proceso ya existía.
- *quita_menu()*. Quita la imagen de fondo del menú.

Creando las nuevas posibilidades

Para añadir las nuevas posibilidades definiremos una estructura *(dialogo)* que podrá contener tres líneas de texto con sus respectivos códigos identificadores e *id* de texto. El código identificador nos servirá para actuar en consecuencia a la respuesta del protagonista.

Cada vez que se entre en una conversación deben presentarse varias frases

El *id* texto nos servirá para borrar sólo esa línea de la pantalla. Cuando el protagonista o el personaje habla, es necesario producir un retardo acorde con lo que va a decir. Esto se controla mediante la constante *r_habla*, que no es más que la tasa de FRAME para el proceso *habla*. La duración del texto en pantalla será diez veces la longi-







tud de la cadena de texto: *FOR (i=0; i< ascii_len(texto)*10; i++) FRAME(r_habla); END*. Nuestro código fuente está preparado para funcionar perfectamente con DIV Games Studio v1.03b, que aún no tenía capacidades para el tratamiento de cadenas. El cálculo de longitud de cadena se hace mediante la función *ascii_len(cadena_de_texto)* de la librería ascii.dll de Antonio Marchal (Tizo). Esta librería debe estar instalada en el directorio dll de DIV.

Más adelante perfeccionaremos mucho más el proceso *habla*; por ahora nos basta con indicar que estamos *hablando* (con un flag hablando), escribir el texto, retardo e indicar que hemos dejado de hablar.

Es interesante tener al menos cuatro frases donde poder elegir

Usaremos *habla()* para los textos del protagonista y *habla2()* para los textos de otros personajes. Las funciones son idénticas, salvo por el tipo de letra que utilizan. Tendremos una variable global *frase* muy peculiar. Cuando *frase* valga –2 nos indicará que no hay opciones de diálogo en curso. Al valer –1 controlaremos que aún no se ha seleccionado una de las opciones. Si tomase otro valor, indica el número de la opción (0 para la primera, 1 para segunda, 2 para la tercera y 3 para la cuarta). Los nuevos procesos que necesitamos crear son:

– *dialoga(uno_t, uno_v, dos_t, dos_v, tres_t, tres_v, cuatro_t, cuatro_v)*. Toma los cuatro textos alternativos del diálogo y su código correspondiente. Su trabajo consiste en actualizar la estructura *dialogo* y hacer pausa si se está hablando en este instante, mientras no se elija opción controlar la posición del ratón sobre las opciones, seleccionándolas o ejecutándolas. Una vez hecha la selección, quita las opciones de pantalla. Al pasarle los parámetros, dejaremos en blanco los que no existan, asegurándonos que al menos debe haber uno.
– *sel_dialogo(opcion)*. Resalta la opción seleccionada cambiando su fuente de letra.
– *haz_dialogo(opcion)*. Comprueba que la opción es válida (no está en blanco) y si es así la toma.

Todo listo

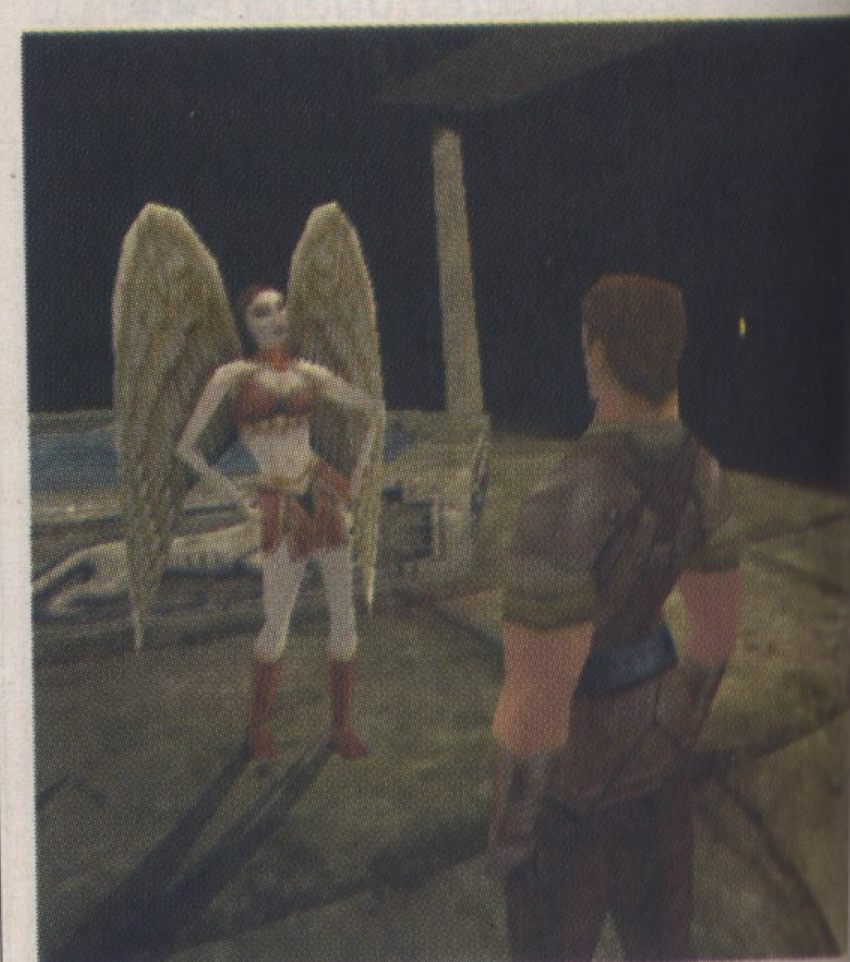
Ya podemos empezar. En aquellos objetos a los que queramos dotar de capacidades de diálogo nos basta con añadirles en su apartado *o_Hablar* el siguiente comportamiento:

Aseguramos que no hay diálogo (si hubiese otro, no se había llegado aquí)
Bucle
Si se está hablando ahora, esperamos
Si no hay diálogo, iniciamos con dialoga (desactiva menú)
Si hay diálogo, pero no se ha seleccionado nada aún, esperamos
Si ya se eligió una opción, actuamos en consecuencia (al menos una debe obligar a salir de este bucle)
Fin del bucle
Aseguramos que no hay diálogo (este ya ha acabado)
Activamos el menú

En nuestro fuente de ejemplo, esto queda así. Código insertado en el objeto diana para dotarlo de capacidades de diálogo:

```
CASE o_Hablar: frase=-2;
  LOOP
    WHILE (hablando)
      FRAME;
    END;
    SWITCH (frase)
      CASE -2:
        dialoga(ascii_add(ascii_add
("Hola, eres una ", nombre), "?"), 1,
            ascii_add("Pareces una ",
nombre),                          2,
            ascii_add("No me gustan
las", nombre),                    3,
            "Adios",              0);
      END;
      CASE -1:
        FRAME;
```



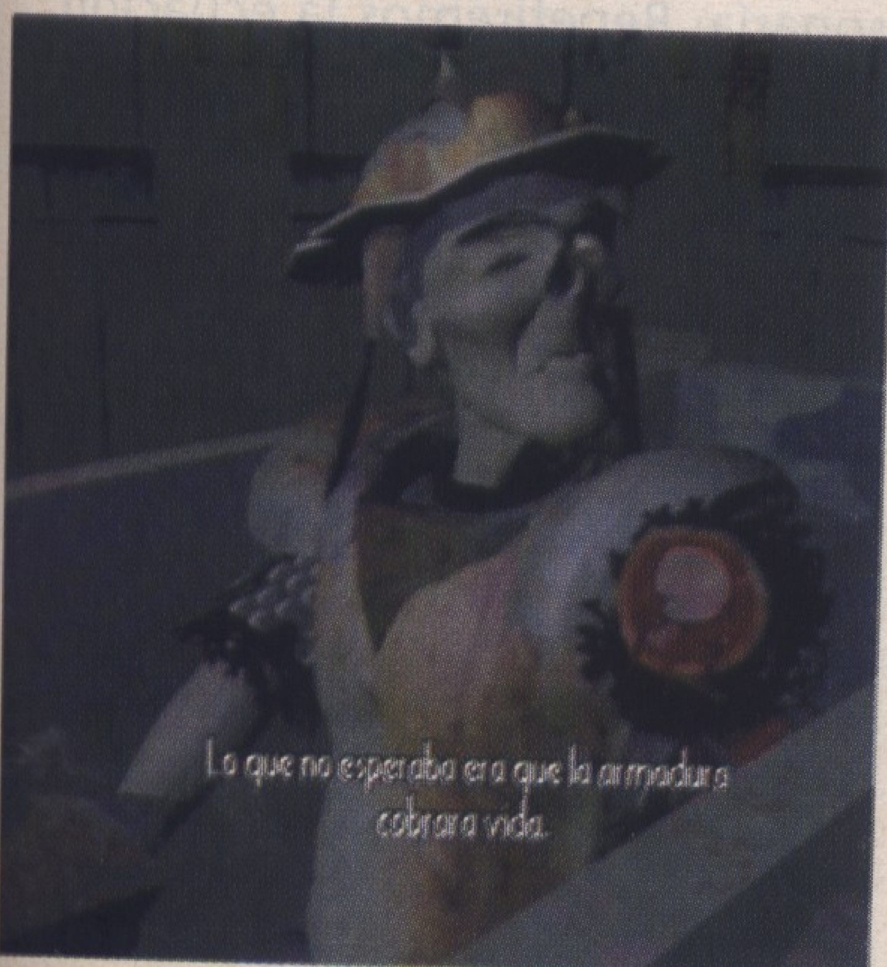




```
END;
    DEFAULT:
     SWITCH
(dialogo[frase].valor)
      CASE 0: habla2("Ciao");
break; END;
      CASE 1:
       habla2("Puede que si,
puede que no");
      END;
      CASE 2:
       habla2(ascii_add("Hasta
puede que sea ", nombre));
      END;
      CASE 3:
       habla2(ascii_add(ascii_add
("Todas las ", nombre),
"no se comen"));
      END;
     END
     frase=-2;
    END;
   END;
  END;
  activa_menu();
 END;
```

En el próximo artículo mejoraremos las capacidades de diálogo de nuestros personajes y haremos conversaciones más complejas. Los ficheros fuente del CD están preparados para ser instalados en el directorio \div\cag\06 en la unidad donde tengas instalado DIV Games Studio. Hasta pronto.

Miguel Adolfo Barroso
(mabarroso@jet.es)



Parte del código del programa original preparado para las nuevas funciones de diálogo. Podéis encontrarlo entero en el CD-Rom que acompaña a la revista.

```
GLOBAL
  id_btn_bolsa_arriba;                    // id boton
  id_tbn_bolsa_abajo;                     // id boton

  menu;                        // Estado actual del menu
  amenu;                       // Estado anterior del menu

 menu = TRUE;
 amenu=menu;

  // Bucle principal
  LOOP
   IF (key(_esc))   // Se ha pulsado 'escape'
     BREAK;        // Se sale del bucle
   END

   // Las teclas cusores desplazan el puntero del rat_n
   IF (key(_right)) mouse.x+=10; END;
   IF (key(_left))  mouse.x-=10; END;
   IF (key(_up))    mouse.y-=10; END;
   IF (key(_down))  mouse.y+=10; END;
   // La barra espaciadora y Return simulan los botones del rat_n

   IF (key(_space)) mouse.left=TRUE; END;
   IF (key(_enter)) mouse.right=TRUE; END;

   IF (menu != amenu)
    IF (menu)
     pon_menu();
     bolsa_muestra();
    ELSE
     quita_menu();
     bolsa_oculta();
     sel_accion(o_none, FALSE);
    END;
    amenu = menu;
   END;

  IF (menu)
   // *** Principio de control del menú de opciones *** //
 *** Bloque de código ***
   // *** Fin de control del menú de opciones *** //
 END;
 FRAME;              // Pasa el control al resto de procesos en ejecuci_n
 END;

 PROCESS pon_menu();
  BEGIN
   put_screen(f_menu, 1);
  END

 PROCESS quita_menu();
  BEGIN
   clear_screen();
  END

 PROCESS activa_menu();
  BEGIN
```

Programación de Juegos de Estrategia - 6

Métodos alternativos de programación

En este número de la revista vamos a hacer una pausa en el desarrollo del juego en el que hemos trabajado hasta ahora, para ver otras formas diferentes de programar algunas cosas que ya se han visto en los números anteriores, y que pueden ser de utilidad para determinadas situaciones.

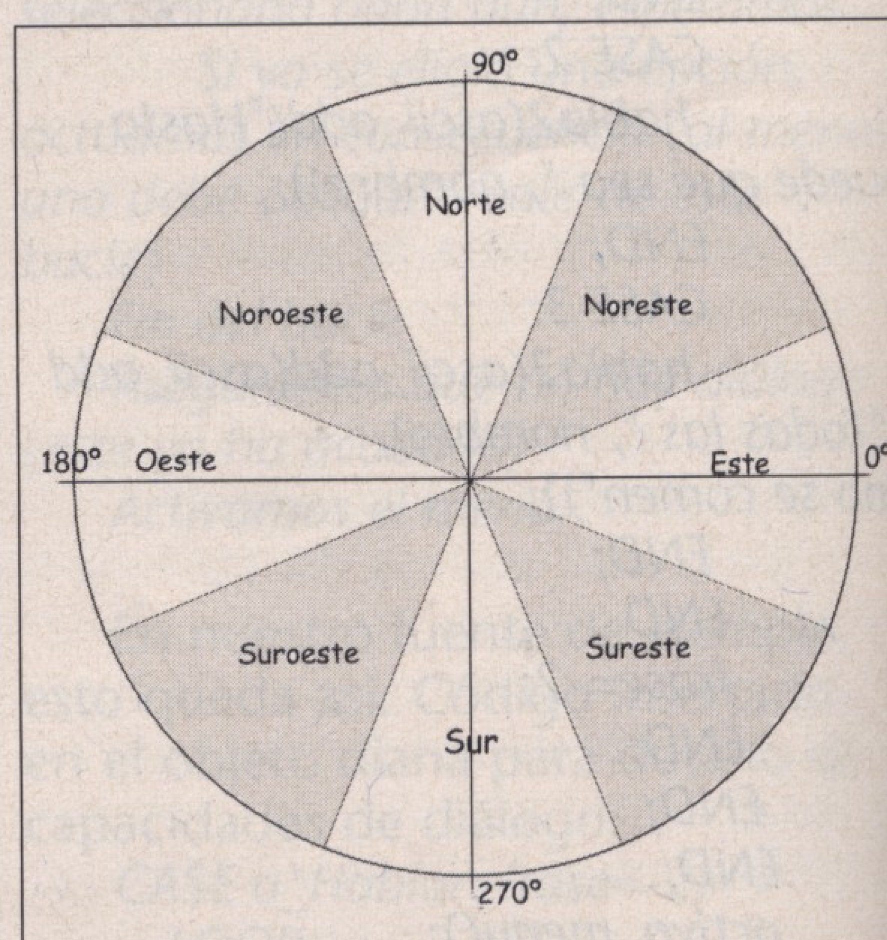
Lo primero que trataremos en el artículo será una forma de representar a los mobs cuando la vista elegida no es la cenital, es decir, no se ven desde arriba. También veremos como simular un scroll sobre un mapa gigante usando tan solo un mapa de tamaño 3x3xTamaño_de_la_region_de_visualizacion. Veremos una forma diferente de implementar menús usando señales y un proceso diferente para cada menú. Para terminar veremos una alternativa al manejo de arrays para simular matrices usando structs.

El problema de las vistas en perspectiva no cenital es que se necesita un gráfico diferente para cada posible posición del objeto

Vistas en ángulo no cenital

El problema de las vistas en perspectiva no cenital es que se necesita un gráfico diferente para cada posible posición del objeto, lo que complica el manejo de los gráficos (ya no es tan sencillo como rotar el gráfico sobre su eje de giro). Esta característica nos obliga a limitar el número de vistas posibles, ya que cada una de ellas ocupa espacio, tanto en disco como en memoria RAM. Lo normal es limitar las posibles posturas de los mobs a 8 (arriba, abajo, derecha, izquierda y las 4 diagonales). Para ahorrar memoria se guardan solo 5 (arriba, abajo, izquierda y las diagonales hacia la izquierda) y se obtienen las otras 3 espejando el gráfico (¿nadie se ha preguntado nunca por qué los orcos y demás bichejos del Warcraft no paran de cambiarse el arma de mano según hacia donde miran como si fueran ambidiestros?). Para elegir qué gráfico pintar, tan solo hay que mirar en qué ángulo está "mirando" el objeto. Lo más sencillo es asociar un rango de ángulos a cada posición como se muestra en la *figura 1* y comprobar en qué rango se encuentra el ángulo del objeto mediante varios ifs consecutivos.





Simulación de matrices mediante structs

Hasta el momento se han usado arrays unidimensionales para simular matrices, haciéndose necesario convertir mediante ecuaciones las coordenadas cartesianas (x, y) a un índice del array. El cálculo es el mismo que debe hacerse al manejar la memoria de vídeo, así que a algunos ya le sonaría. Repetiremos la ecuación para los despistados:

*Indice = x + (y * MAX_X).*

El problema es, que es poco intuitivo, así que se puede optar por la alternativa de codificarlas de una forma que, aunque dé acceso más lento, tiene una sintaxis mucho más clara. Si queremos realizar una matriz de 2 dimensiones como la que simulábamos con el array, tan solo tenemos que definir una estructura como la que se muestra a continuación:



matriz_x[0]	matriz_x[1]	matriz_x[2]	matriz_x[3]	matriz_x[4]
y[0]	y[0]	y[0]	y[0]	y[0]
y[1]	y[1]	y[1]	y[1]	y[1]
y[2]	y[2]	y[2]	y[2]	y[2]
y[3]	y[3]	y[3]	y[3]	y[3]
y[4]	y[4]	y[4]	y[4]	y[4]
y[5]	y[5]	y[5]	y[5]	y[5]

```
struct matriz_x[100]
    y[100];
end
```

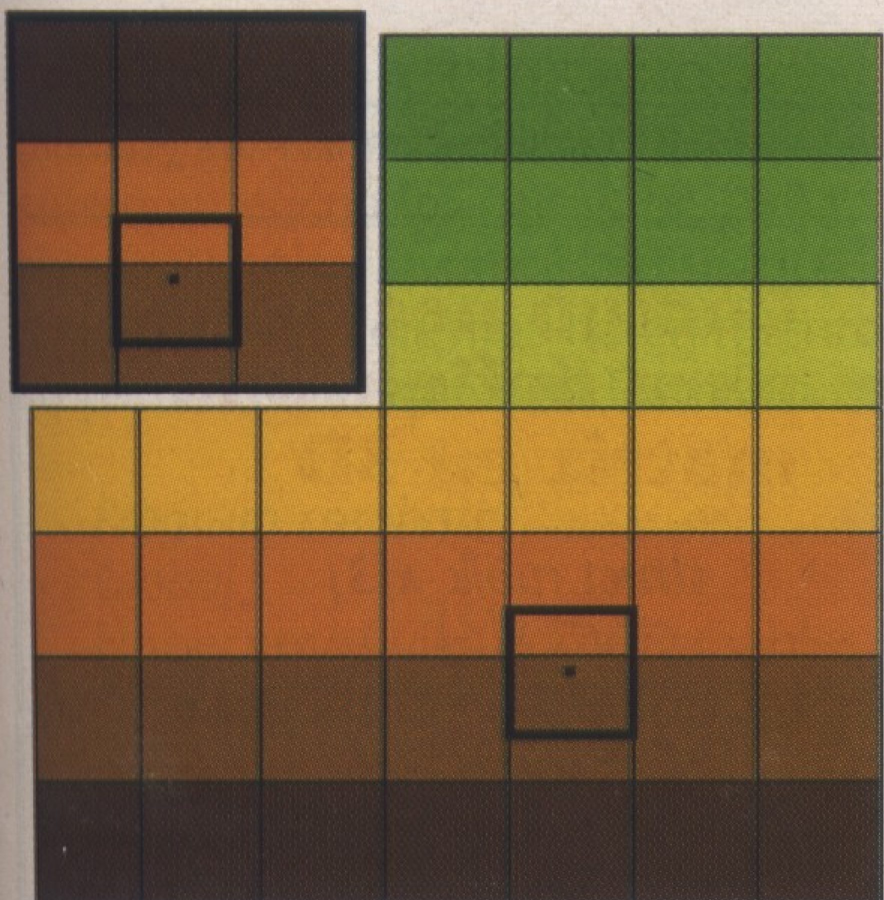
y luego acceder a la misma en cualquier lugar del código de la siguiente manera:

```
matriz_x[i].y[j];
```

Si se necesita alguna vez una matriz de más dimensiones, se puede hacer de forma muy sencilla con solo definir otra estructura bastante similar pero con algunos cambios (en realidad se define un array de matrices bidimensionales, lo cual es lo mismo que una matriz de tres dimensiones):

```
struct matriz_x[100]
    struct y[100]
        z[100];
    end
end
```

y acceder de manera parecida poniendo *matriz_x[i].y[j].z[k]*. Los programadores que utilicen el DIV II se preguntarán por qué complicarse la vida cuando pueden definirse matrices de manera muy simple. Aclarar ante todo que hasta el momento todo lo que aparece en los artículos es orientado a la programación con DIV I, para que pueda ser aprovechado por la mayoría de los usuarios, por lo que



algunos truquillos no serán necesarios para los programadores que usen el DIV II. En la *figura 2* podemos observar la representación de una matriz realizada con estructuras anidadas. En ella se observa como en realidad lo que hacemos es un array de arrays. Y ahora sigamos con el siguiente truquillo.

Menús durmientes

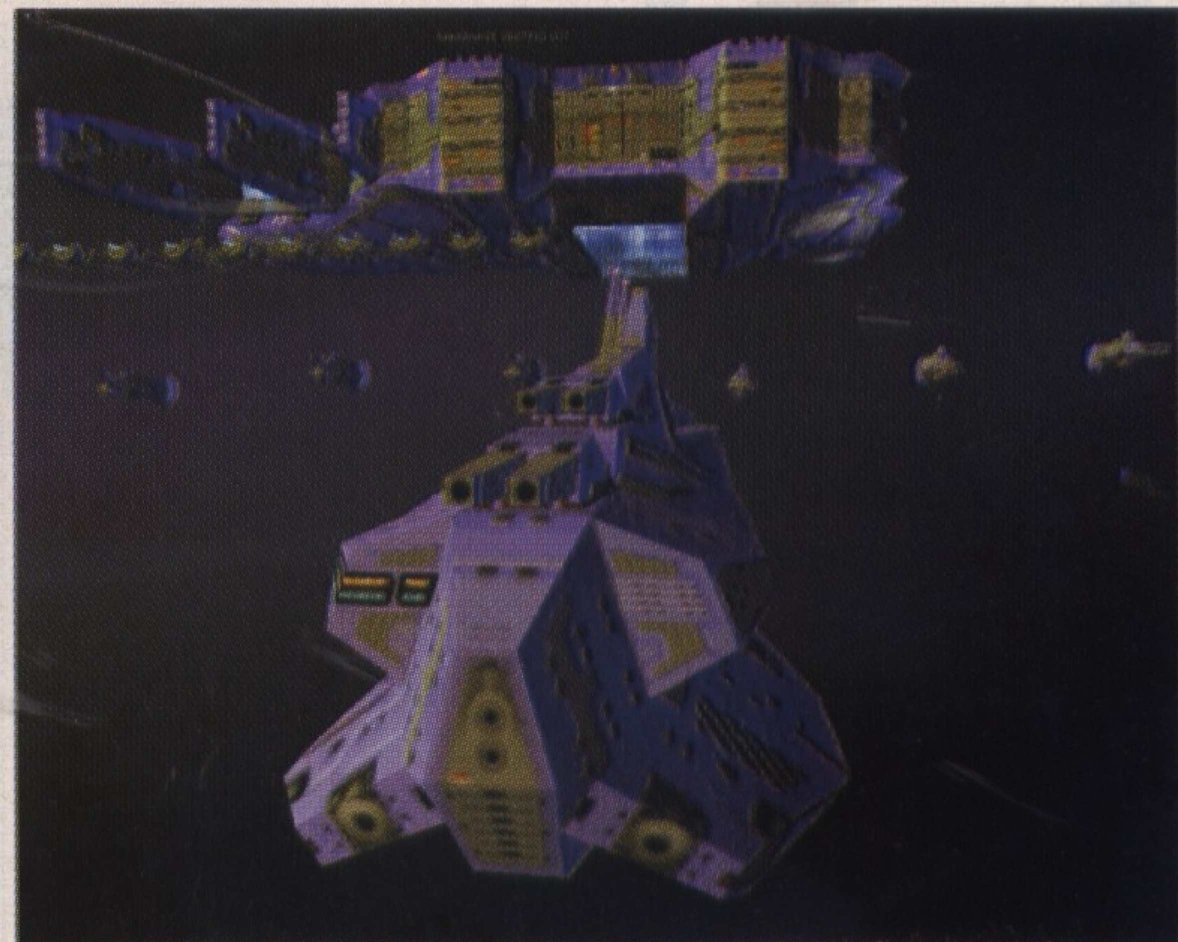
La estructura utilizada para representar los menús, en el ejemplo que acompaña todos los artículos, está implementada sobre un único proceso separado en secciones de código según el menú que deba aparecer, manteniéndose en cada sección por medio de un bucle condicional y varios ifs para comprobar si debemos cambiar de estado. El sistema está inspirado en un autómata determinista, aunque eso es otra historia que no nos atañe en esta sección. Si se desea utilizar un método alternativo de representar menús, se puede implementar cada menú por separado en procesos independientes, llamando desde el proceso principal al menú raíz. Cada submenú será llamado por este menú, y cada uno de estos llamará a sus submenús. Para que no se superpongan unos menús sobre otros, cada menú debe dormir a su padre y a todos sus hijos excepto a sí mismo, para lo que se realiza la siguiente secuencia de instrucciones:

```
signal(father,s_sleep_tree);
    signal(id,s_wakeup);
```

La primera instrucción manda una señal al padre del proceso (el proceso que lo llamó) indicándole que se duerma él y que propague la señal a todos sus hijos. El problema es que nos dormiríamos nosotros también, por lo que debemos mandarnos una señal s_wakeup para despertarnos y así evitar un bloqueo. Cuando termine el proceso, debe mandar de nuevo una señal a su padre para que se despierten tanto él como sus hijos. Este sistema consume más recursos que el método de un solo proceso, pero es mucho más sencillo de programar y ampliar.

Scroll sobre mapa virtual

Este "truco" es de nivel avanzado, pero puede resultar de mucha utilidad si queremos representar un mapa de juego de grandes proporciones (por ejemplo un mapa de 1000x1000 casillas de juego, cada una de las cuales de 32x32 pixeles y que ocuparía 976'56Mb) sin



necesidad de requerir un ordenador con 1Gb de RAM. Con el sistema utilizado (manejo de map_put sobre una imagen vacía de las dimensiones necesarias) necesitaríamos una imagen base de varios Mbytes para representar un mapa medianamente decente, el cual debe cargarse en memoria completamente, incluso si solo se pinta en pantalla una pequeña proporción del mismo. Partiendo de esta idea se puede pensar en tener solo una parte del mapa cargado en memoria, e ir pintando sobre la marcha lo que debe verse en pantalla. Es decir, tener una imagen base más pequeña y una matriz con el mapa completo representado por los índices de los *tiles* que deben pintarse. Según se vaya desplazando el scroll por el mapa, iremos pintando en la imagen base los *tiles* que deberán aparecer a continuación aprovechando el scroll cíclico en todas las direcciones. Aquí aparece el primer problema, ya que si la imagen base es muy pequeña, el pintado de los *tiles* deberá ser muy frecuente, ralentizando el scroll de manera considerable. Tenemos que elegir un tamaño de imagen base que no implique repintar cada vez que avanzamos el scroll, pero que no ocupe demasiado en memoria. La opción más asequible es tener una imagen base de 3 veces el alto de la región de visualización por 3 veces el ancho. Así, dividiremos la imagen base en tres bandas horizontales y tres verticales (conformando una matriz de 9 casillas) que numeraremos del 0 al 2. Supongamos que el mapa total tendría un tamaño en regiones de visualización de 10 por 10. Si el centro de la región de visualización se mantiene en las bandas de los bordes (horizontales y verticales) no repintaremos, pero en cuanto se adentre en otra región deberemos comprobar que tenemos pintadas las bandas veci-

Hasta el momento se han usado arrays para simular matrices

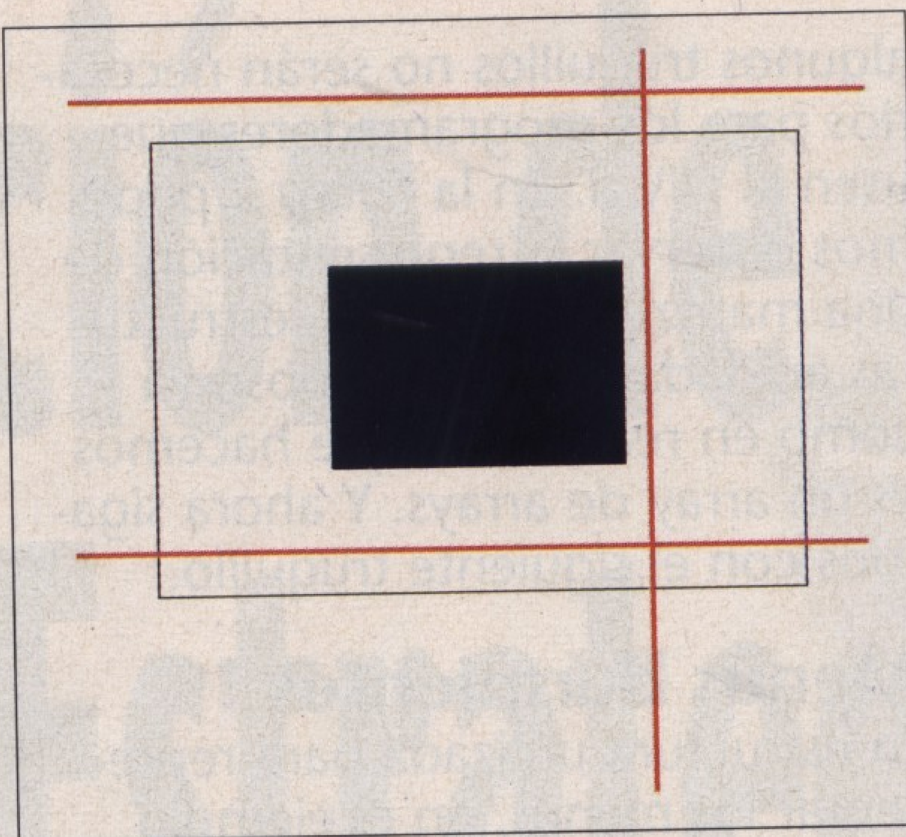
nas en la imagen base. Por ejemplo, si nos adentramos desde la banda horizontal 1 a la banda 2, debemos tener pintadas las bandas horizontales 1, 2 y 3, pero tenemos las bandas 0, 1 y 2. ¿Cómo sabemos qué banda debemos sustituir por la 3? Pues aprovechando las propiedades del modulo, es decir calculando el modulo 3 de las bandas que debemos tener pintadas:

- **1 mod 3** = 1 (la pintaríamos en la 1, pero ya está)
- **2 mod 3** = 2 (la pintaríamos en la 2, pero ya está)
- **3 mod 3** = 0 (la pintamos en la 0)

Fácil ¿no?. En la *figura 3* se muestra la imagen base sobre el mapa que simulamos. Se ve cómo la fila marrón oscuro se pinta en la banda 0 horizontal (fila 6 mod 3 = 0), mientras que las filas naranja y marrón claro se pintan en las bandas 1 y 2 (filas 4 y 5 mod 3 = 1 y 2). Hasta el momento hemos mantenido un nivel de abstracción bastante alto manejando bandas de memoria. Vamos a empezar a ver qué datos debemos controlar y mantener guardados para que todo salga bien.

Para gráficos de 60x60 la función de colisión del DIV hace 3600 comprobaciones cada vez

Lo primero que debemos conocer es el ancho en casillas de una banda vertical y el alto de una banda horizontal, luego debemos guardar qué banda virtual está en cada una de las tres bandas reales



(tanto horizontales como verticales). También debemos tener una forma de calcular en qué banda virtual está la región de visualización para poder comprobar si es necesario repintar o no (una posible mejora del algoritmo sería hacer que comprobara la banda en la que está con un rectángulo y no con un solo punto, es decir que no considerara cambio de región hasta que todo un recuadro haya traspasado la barrera: ver *figura 4*).

Diferentes tipos de colisión

La colisión usada en el juego del curso está basada en la colocación de los personajes en casillas, lo que simplifica mucho el algoritmo, pero si hacemos un juego sin rejilla, deberemos usar algún otro tipo de detección de la colisión. Ahora veremos algunas opciones. La más obvia es usar la propia que trae el DIV, pero si tenemos muchos mobs simultáneos es bastante lenta, ya que comprueba la superposición de los pixeles que conforman las dos imágenes que se están comprobando. Si los gráficos son pequeños, no hay tanto problema, pero imaginad dos gráficos de 60x60 pixeles cada uno (¿no son tan grandes verdad?): pues la colisión del DIV tendría que hacer 3600 comprobaciones por cada par de gráficos que se comprobaran. Si no necesitamos precisión, podemos optar por la colisión por cuadrados, que tan solo comprueba si los cuadrados que engloban cada gráfico se superponen (4 comprobaciones por cada par de gráficos). Este tipo de detección de colisión sólo comprueba si alguna de las esquinas del cuadrado de un gráfico está dentro del cuadrado del otro gráfico. Otra opción es la detección esférica, muy utilizada en simuladores y juegos en 3D, y que se basa en considerar los objetos como esferas de un radio determinado. Con este método solo debe hacerse una comprobación, aunque más compleja en los cálculos que las anteriores. Tan solo hay que comprobar que la distancia que separa los centros de los objetos no sea menor que la suma de sus radios. La ventaja de este sistema es que resulta mejor en juegos con vista en 3D, ya que no importa por donde choquen, el choque es igual. El problema es que el calculo de la distancia entre los dos objetos es algo lento ya que entran en juego funciones trigonométricas. La mejor opción, como casi siempre, es la combinación de varias. Usar primero la colisión por cuadrado, y si ésta nos indica que hay colisión entonces afinar con la colisión por pixeles (la de DIV).

Para despedirnos

Por el momento no tenemos más trucos bajo la manga, así que en el próximo número continuaremos programando el juego que nos ha ocupado hasta hoy. Todavía hay que terminar el parser, codificar la IA y alguna que otra cosilla que ya veremos en próximos números.

Emilio Llamas Alba (YuMoK)

Código de los procesos que nos indican qué gráfico pintar según hacia donde mire

```
PROCESS calculate_graph(posicion)
BEGIN
  if(posicion < 0)
    posicion *= -1;
    father.flags = father.flags or 1;
  else
    father.flags = father.flags and (32|16|8|4|2) ;
  end
  father.graph = posicion +
          mobs[father.number].tipo_bicho + 1;
END


PROCESS en_que_posicion_estoy(angulo,p_posicion)
BEGIN
  if(angulo <= 22500 && angulo > -22500) *p_posicion=East; end
  if(angulo <= -22500 && angulo > -3*22500) *p_posicion=SEast; end
  if(angulo <= -3*22500 && angulo > -5*22500) *p_posicion=South; end
  if(angulo <= -5*22500 && angulo > -7*22500) *p_posicion=SWest; end

  if(angulo <= -7*22500 && angulo > 7*22500) *p_posicion=West; end
  if(angulo <= 7*22500 && angulo > 5*22500) *p_posicion=NWest; end
  if(angulo <= 5*22500 && angulo > 3*22500) *p_posicion=North; end
  if(angulo <= 3*22500 && angulo > 22500) *p_posicion=NEast; end
END
```

Código necesario para definir una matriz en 3D

```
struct rubik_x[3]
  struct y[3]
    z[3];
  end
end
```

Programando un juego de rol

Sistemas de combate

En este artículo hablaremos de los sistemas de combate en un juego de rol. Esta es una de las partes más importantes de cualquier RPG, ya que uno de los fundamentos de los juegos de rol es luchar para conseguir más experiencia y así mejorar nuestras habilidades y características.

No todos los sistemas de combate son iguales, pero podríamos clasificarlos en dos grandes grupos:

Acción directa: En estos sistemas de combate el jugador lucha directamente con los enemigos, sin que se produzca ninguna pausa en el desarrollo del juego. Algunos ejemplos de este tipo de sistemas de combate podrían ser *Zelda*, *Secret of Mana* o *Diablo*, por citar algunos.

Acción parada: En estos sistemas de combate al encontrarse con un enemigo se para la acción (o incluso se cambia de escenario) para poder empezar la lucha. El combate puede desarrollarse por turnos *(Breath of Fire)* o bien a tiempo real *(Final Fantasy, Chrono Trigger)*. Estos últimos son los mejores ya que no permiten al jugador pararse a pensar, consiguiendo así un combate más fluido y realista.

A partir de ahora nos centraremos, analizando un ejemplo, en el último sistema de combate comentado, el de acción parada a tiempo real.

Explicación del código

Este código lo podéis encontrar entero en el cuadro que acompaña al artículo. Ahora vamos a diseccionarlo para ver con más detenimiento las partes de las que está formado.

```
Program combate;
Global
```

Empecemos viendo las variables y las estructuras del ejemplo:

```
    struct menuc
      op=2;
      textos[10]="Atacar","Escapar";
    end

    struct comba
      ne=3;

      struct enemigo[2]
        cont;
        byte listo;
        bap=4;
        aap=5;
        bdp=3;
        adp=3;
        as=45;
        ds=40;
        luc=30;
        hp=15;
        mp=0;
        x;
        y;
      end

      struct jugador
        cont;
        byte listo;
        estado;
        x;
        y;
        hp=100;
        mp=10;
        bap=5;
        aap=6;
        bdp=4;
        adp=5;
        as=55;
        ds=50;
        luc=40;
      end

    end

    byte atacando;
    target;
```



En Breath of Fire 2 los combates se desarrollan en acción parada y por turnos.



Estructura menuc: esta estructura define el número de opciones que tendrá el menú (variable *op*) y los textos de las mismas, que se guardan en la tabla *textos*.

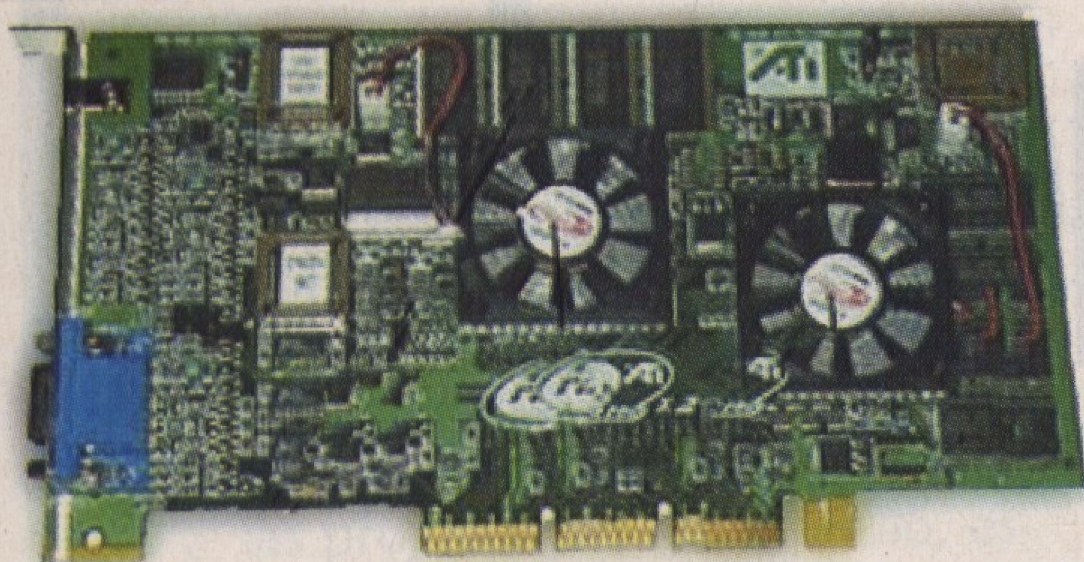
Estructura comba: es la estructura principal del combate, en ella definiremos el número de enemigos (variable *ne*) y las características de los enemigos y el jugador.

Subestructura enemigos y jugador: aquí definiremos todas las variables que van a usar los enemigos o el jugador:

Variables de posición *(x,y)*. Indican las coordenadas dónde se encuentra el enemigo o el jugador.

Variables de las características del jugador o de los enemigos *(hp,mp,aap,bdp,as,ds,luc)*. Indican la fuerza, suerte, defensa y otros parámetros.

Variable cont. Este contador se usa para saber cuándo el enemigo está listo para atacar.



Variable listo. Esta variable determina si se está listo o no para atacar.

Variable estado. Según el valor de esta variable se realizarán diferentes acciones.

Variable atacando. Si esta variable está a 1 se indica que en estos momentos hay alguien atacando; por lo tanto, habrá que esperar a que se acabe de atacar.

Variable target. Indica el objetivo al cual se va a atacar.

```
Begin
    comba.enemigo[0].x=400;
    comba.enemigo[0].y=280;
    comba.enemigo[1].x=500;
    comba.enemigo[1].y=90;
    comba.enemigo[2].x=300;
    comba.enemigo[2].y=400;
    comba.jugador[0].x=200;
    comba.jugador[0].y=180;

    load_fpg("combat.fpg");
        set_mode(m640x480);
        combat();

    End
```

Los combates son una de las partes más entretenidas de los juegos de rol

Inicializamos algunos datos de las estructuras, cargamos el *fpg*, ponemos el modo gráfico y llamamos al proceso *combat*.

```
Process combat()

Begin
    menu();
    player();
    enemigo(0,400,280);
    enemigo(1,500,90);
    enemigo(2,300,400);

End
```

Este proceso llama a todos los procesos que formarán el combate, que son el del menú, el del jugador y a 3 enemigos.

```
Process menu();

Private
    i;
Begin
    graph=1;
    x=320;
    y=430;

    for(i=0;i<menuc.op;i++)

write(0,50,10*i+400,0,menuc.textos[i
]);
    end

    cursor();

    Loop
        frame;
    End

End
```

El proceso menú escribe los textos según los parámetros de la estructura *menuc*, pone el gráfico del menú y llama al proceso *cursor*.

```
Process cursor();

Private
pos=0;
jugid=0;

Begin

graph=2;
size=25;
x=30;
y=400;

Loop
  if(key(_down))
    pos++;
  end

  if(key(_up))
    pos--;
  end

  if(pos>menuc.op-1)
    pos=0;
  end

  if(pos<0)
```



Zelda para Super Nintendo. Los combates son de acción directa.




```
    pos=menuc.op-1;
  end

  y=400+(10*pos);

  if(key(_enter) &&
comba.jugador[jugid].listo==1)

    switch(pos)

      case 0:
        target=select_ene();
        esta_listo(&atacando);
        comba.jugador[0].esta-
do=1;
        esta_listo(&comba.juga-
dor[jugid].estado);
        comba.enemigo[target]
.hp-=ataque(comba.jugador[jugid]
.bap,comba.jugador[jugid].aap,comb
a.jugador[jugid].as,comba.jugador[ju
gid].luc,comba.enemigo[target].bdp,c
omba.enemigo[target].adp,comba.ene
migo[target].ds,comba.enemigo[tar-
get].luc,comba.enemigo[target].x,com
ba.enemigo[target].y);
        atacando=0;
      end

      case 1:
        esta_listo(&atacando);
        comba.jugador[0].esta-
do=2;
      end

    end

  comba.jugador[jugid].listo=0;

  comba.jugador[jugid].cont=0;
  end

    frame;
  End

End
```

Este proceso se encarga de desplazar el cursor del menú según las pulsaciones recibidas desde el teclado. Si se presiona la tecla "enter" y la variable *listo* es igual a 1 se elegirá una acción según el valor de la variable *pos* (que indica la posición del cursor en el menú).

```
function esta_listo(var)
Begin
   repeat
      frame;
   until(*var==0)
End
```

Esta función resulta muy útil, ya que nos permite saber si una variable es 0. Saber si una variable es 0 es, en nuestro caso, necesario para saber si hay alguien atacando. Esta función recibe como parámetro la dirección de la variable, y después comprueba si el contenido de esa dirección es 0 o no.

```
function ataque (bap,aap,as,luc,
bdp,adp,ds,luc1,x,y)

Private

   atack;
   defense;

Begin
      atack= bap + rand(0, (
(aap*as)/100 ) );

      if(rand(0,100)<luc)
         atack+=
(aap*rand(0,100))/100;
      end

      switch(rand(0,100))

      case 0..2:
         atack=0;
      end

      case 98..100:
         atack=bap+aap;
      end
      end

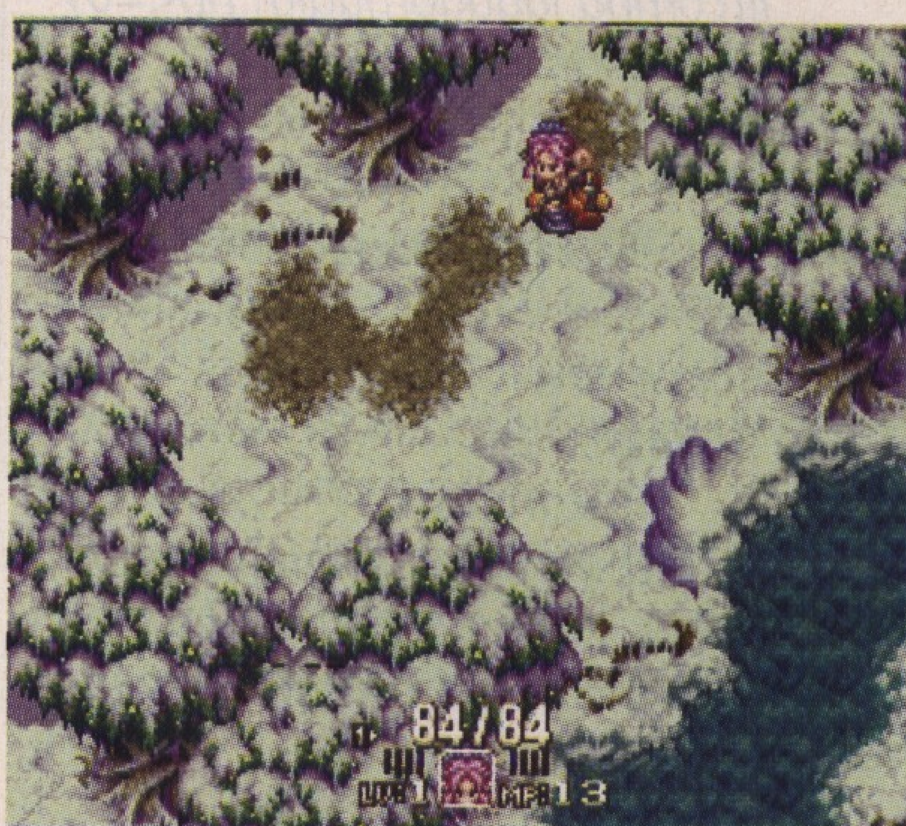
      defense= bdp + rand
(0, ( (adp*ds)/100 ) );

      if(rand(0,100)<luc1)
         defense+= (adp*rand
(0,100))/100;
      end

      switch(rand(0,100))
```



Final Fantasy 6, un juego donde los combates son del tipo acción parada en tiempo real.




```
      case 0..2:
         defense=0;
      end

      case 98..100:
         defense=bdp+adp;
      end
      end

      atack-=defense;
      da o(x,y,atack);

      if(atack>0)
         return(atack);
      else
         return(0);
      end

   End
```

Esta función es la encargada de determinar el daño que se le causará al enemigo. Recibe como parámetros las características del atacante y las del defensor, para poder establecer el ataque en función de ambos contendientes en vez de hacerlo aleatoriamente.

```
   process da o(x,y,dam);

   Private
      i;
      texto;

   Begin
      texto=write_int(0,x,y,0,&dam);

      for(i=0;i<25;i++,y-=2);
         delete_text(texto);

         if(dam>0)

texto=write_int(0,x,y,0,&dam);
         else
            texto=write(0,x,y,0,"MISS");
         end

         frame;
      end

      delete_text(texto);
   end
```

Este proceso muestra el daño causado, aplicándole una pequeña animación de movimiento hacia arriba.

```
function sel_pos(pos,mode);
Private
   lastpos;
Begin
   lastpos=pos;
   repeat
      switch(mode)
         case 0:
            pos++;
            if(pos>comba.ne-1)
               pos=0;
            end

            if(pos==lastpos)
               break;
            end
         end

         case 1:
            pos--;
            if(pos<0)
               pos=comba.ne-1;
            end
            if(pos==lastpos)
               break;
            end
         end
      end

until(comba.enemigo[pos].hp>0)
   return(pos);
End
```

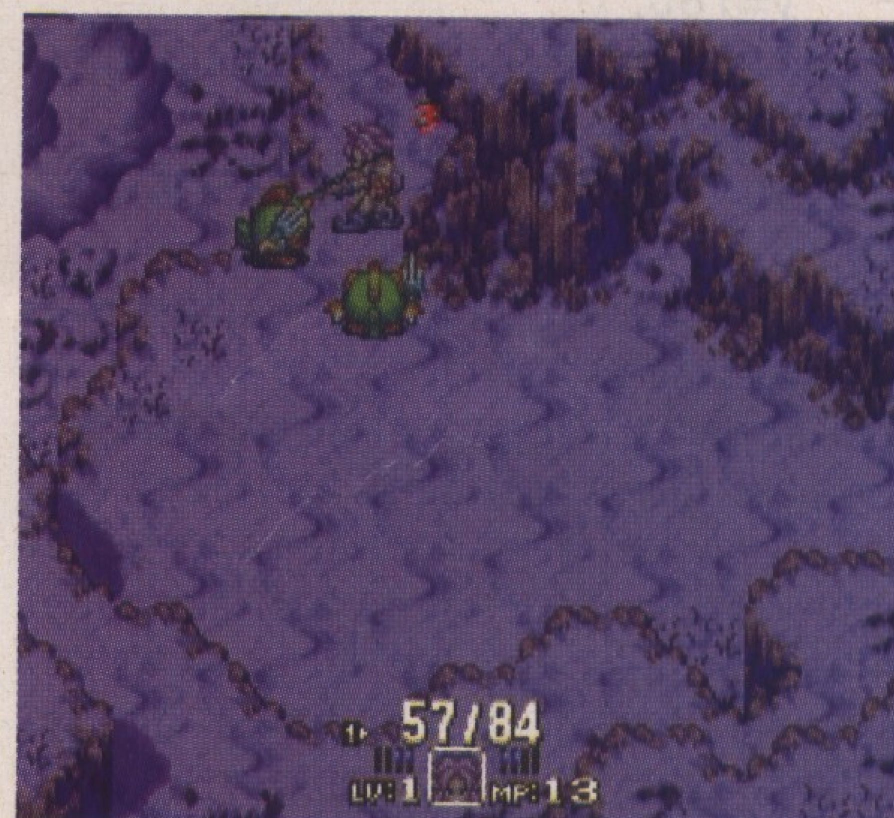
Esta función es llamada por la función *select_ene* y permite saber cuál es el siguiente enemigo (indistintamente del número de enemigos, siempre que las estructuras estén con los datos correctos) según hayamos presionado arriba o abajo.

> El tipo de combate que usarán los protagonistas de nuestro juego es algo que tendremos que decidir al principio

```
function select_ene()

Private
pos;
lastpos;

Begin
   size=25;
   graph=2;
   z=-100;
```



Imágenes de Secret of Mana 2, en las cuales podemos ver un sistema de combate muy similar al de Zelda.



Combates de acción directa en Diablo2.

```
    if(comba.enemigo[pos].hp<=0)
        pos=sel_pos(pos,0);
    end

    Loop
        if(key(_down))
            pos=sel_pos(pos,0);
        end

        if(key(_up))
            pos=sel_pos(pos,1);
        end

        y=comba.enemigo[pos].y;
        x=comba.enemigo[pos].x;
        if(key(_space))
            return(pos);
        end

        frame;
    end

End
```

Cada modo de combate tiene unas características distintas que deberemos programar

Esta función devuelve el enemigo seleccionado y se encarga de que el cursor que señala al enemigo esté bien posicionado.

```
process player();

Private
    jugid=0;
Begin

    graph=new_map(100,100,50,
50,122);
    x=200;
    y=180;
    write_int(0,200,180,0,&comba.
jugador[jugid].cont);
    write_int(0,200,190,0,&comba.
jugador[jugid].listo);
    write_int(0,240,180,0,&comba.
jugador[jugid].hp);
    write_int(0,240,190,0,&comba.
jugador[jugid].mp);

    Loop
        if(comba.jugador[jugid].
cont<100)
            comba.jugador[jugid].cont++;
        else
            comba.jugador[jugid].listo=1;
        end
        if(comba.jugador[jugid].hp<=0)
            signal(id,s_kill);
        end

        switch(comba.jugador[0].estado)

            case 1:
                loop
                    atacando=1;
                    angle=fget_angle(comba.
jugador[0].x,comba.jugador[0].y,com
ba.enemigo[target].x,comba.enemigo
[target].y);
                    advance(5);

                    if(x>comba.enemigo
[tARGET].x)
                        x=comba.jugador
[jugid].x;
                        y=comba.jugador
[jugid].y;
                        break;
                    end

                    frame;
                end
                comba.jugador[0].estado=0;

            end

            case 2:
                comba.jugador[jugid].
estado=0;

                if(rand(0,3)==0)
                    exit(0,0);
                end
            end
        end

        frame;
    End

End
```

Este proceso es el que mueve el gráfico del personaje que controlamos. Según el valor en que se encuentre la variable *estado* se hará una cosa u otra (0=nada, 1=atacando, 2=intentando escapar).

```
process enemigo(eneid,x,y);

Private

Begin

    graph=new_map(100,100,50,
50,222);

    write_int(0,x,y,0,&comba.
enemigo[eneid].cont);
    write_int(0,x,y+10,0,&comba.
enemigo[eneid].listo);
    write_int(0,x+40,y,0,&comba.
enemigo[eneid].hp);
    write_int(0,x+40,y+10,0,&
comba.enemigo[eneid].mp);

    Loop

        if(comba.enemigo[eneid].
cont<100)  comba.enemigo[eneid].
cont++;
        else
            comba.enemigo[eneid].
listo=1;

            repeat
                if(comba.enemigo
[eneid].hp<=0)
                    signal(id,s_kill);
                    frame;
                end
                frame;
            until(atacando==0)

            if(comba.enemigo
[eneid].hp<=0)
                signal(id,s_kill);
                frame;
            end

            atacando=1;
```



```
loop
    angle=fget_angle
(comba.jugador[0].x,comba.juga-
dor[0].y,comba.enemigo[eneid].x,com
ba.enemigo[eneid].y);
    advance(-5);
    if(x<comba.
jugador[0].x)
        x=comba.enemigo
[eneid].x;
        y=comba.enemigo
[eneid].y;
        break;
    end

    frame;
end

atacando=0;
```

```
comba.jugador[0].hp-
=ataque(comba.enemigo[eneid].bap,c
omba.enemigo[eneid].aap,comba.ene
migo[eneid].as,comba.enemigo[eneid]
.luc,comba.jugador[0].bdp,comba.jug
ador[0].adp,comba.jugador[0].ds,com
ba.jugador[0].luc,comba.jugador[0].x,
comba.jugador[0].y);
    comba.enemigo[eneid].
cont=0;
    comba.enemigo[eneid].
listo=0;
    end

    if(comba.enemigo[eneid].
hp<=0)
        signal(id,s_kill);
        frame;
    end
```

```
    frame;
  End

End
```

Este proceso es muy similar al anterior. Controla las acciones del enemigo.

El código completo de este ejemplo podréis encontrarlo en el CD-Rom que acompaña a la revista, además de en el cuadro adjunto.

Y ya sabéis, si tenéis alguna duda, sugerencia o comentario, hacédnoslo saber. Hasta el próximo número.

Ramón de España
maqnaziller@pagina.de

Cuadro 1. Código de ejemplo

```
Program combate;

Global

  struct menuc
    op=2;
    textos[10]="Atacar","Escapar";
  end

  struct comba
    ne=3;

    struct enemigo[2]
      cont;
      byte listo;
      bap=4;
      aap=5;
      bdp=3;
      adp=3;
      as=45;
      ds=40;
      luc=30;
      hp=15;
      mp=0;
      x;
      y;
    end

    struct jugador
      cont;
      byte listo;
      estado;
      x;
      y;
      hp=100;
      mp=10;
      bap=5;
      aap=6;
      bdp=4;
      adp=5;
      as=55;
      ds=50;
      luc=40;
    end

end

    byte atacando;
    target;

  Begin

    comba.enemigo[0].x=400;
    comba.enemigo[0].y=280;
    comba.enemigo[1].x=500;
    comba.enemigo[1].y=90;
    comba.enemigo[2].x=300;
    comba.enemigo[2].y=400;
    comba.jugador[0].x=200;
    comba.jugador[0].y=180;

    load_fpg("combat.fpg");
    set_mode(m640x480);
    combat();

  End


Process combat()

  Begin

    menu();
    player();
    enemigo(0,400,280);
    enemigo(1,500,90);
    enemigo(2,300,400);

  End


Process menu();


Private
    i;


Begin

    graph=1;
    x=320;
    y=430;

    for(i=0;i<menuc.op;i++)
        write(0,50,10*i+400,0,menuc.
textos[i]);
    end

    cursor();

    Loop

        frame;

    End

End


Process cursor();

Private
    pos=0;
    jugid=0;

Begin

    graph=2;
    size=25;
    x=30;
    y=400;

    Loop
        if(key(_down))
            pos++;
        end

        if(key(_up))
            pos--;
        end

        if(pos>menuc.op-1)
            pos=0;
        end

        if(pos<0)
            pos=menuc.op-1;
```

Una alternativa muy potente

ModPlug Tracker

Aunque resulta difícil de creer, habrá lectores que no habrán encontrado *FastTracker* de su agrado. Para ellos presentamos en este número una alternativa que poco tiene que envidiarle al programa que nos ha mantenido ocupados durante varias entregas.



El primer secuenciador que realmente trabaja bien bajo Windows.

Y para los lectores que hayan caído en las cibernéticas garras de *FastTracker*, les aconsejamos que no pierdan de vista este otro programa de composición musical, ya que si han llegado a manejar con soltura *FastTracker*, no tendrán ningún problema para hacerse dueños y señores del que nos ocupará en este número de DIVManía. Recién salido de Internet, con apenas 4 meses de vida, aquí tenemos un nuevo secuenciador para Windows, *ModPlug Tracker*.

Windows

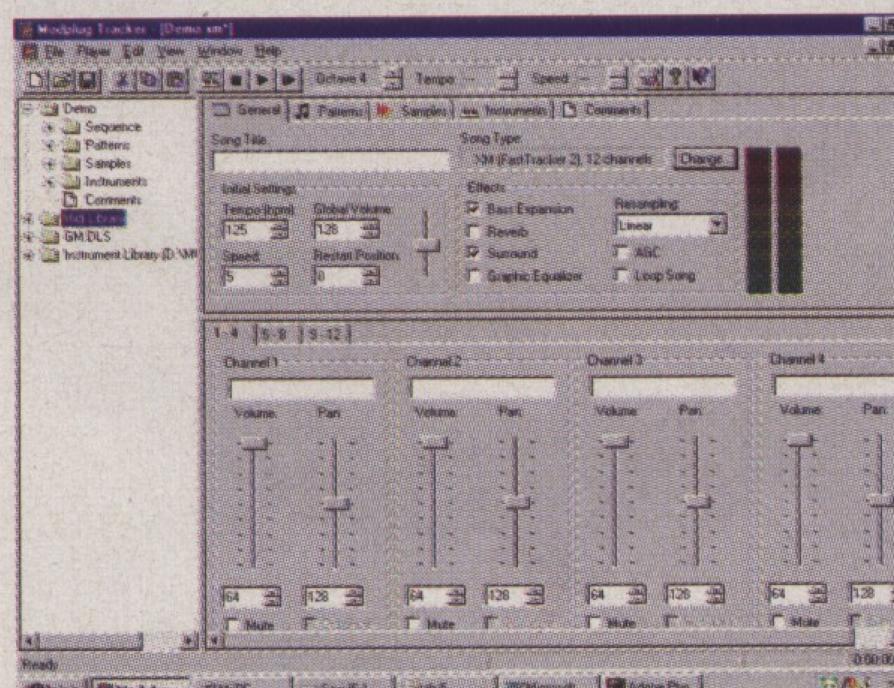
La característica más importante de *ModPlug Tracker* es que está diseñado para trabajar en Windows. Esto comporta algunas ventajas y desventajas con respecto a *FastTracker*. La ventaja principal es que funciona mejor que *FT2* bajo Windows, que, como ya habréis comprobado, sufre considerables retardos cuando es corrido bajo este sistema. Esto no quiere decir que *ModPlug Tracker* no sufra retardos. A veces puede pegar algún "bote" como todos los programas de Windows, sobre todo si el ordenador utilizado no es muy rápido o dispone de poca RAM. En todo caso, la experiencia ha resultado bastante positiva, ya que con un Pentium 150, con 64 Mbytes de RAM, corre muy fluídamente (más que muchos otros programas de Windows diseñados para diversos y distintos propósitos). La desventaja de que el programa sólo se pueda correr bajo Windows es que, por pocos retardos que se sufran jamás nos podremos librar de ellos, y dependeremos siempre de los "caprichos" de un sistema operativo de dudosa eficacia. No se aconseja, entonces, hacer uso de *ModPlug Tracker* para reproducción en casos en que la perfección de ésta sea imprescindible.

ModPlug Tracker ha sido implementado para trabajar únicamente bajo Windows

Otra de las características de *MPT*, heredada del hecho de ser una aplicación para Windows, es su sistema de Cortar/Copiar/Pegar, mucho más intuitivo, por consiguiente, que el de *FT2*. Bastará con seleccionar con el ratón el bloque que deseemos copiar o cortar, pinchar en la opción correspondiente en el menú de la parte superior de la ventana, y después pegar con la misma sencillez. En caso de ser usuarios con más experiencia podremos hacer uso de las combinaciones de teclas Ctrl-X, Ctrl-C y Ctrl-V para cortar, copiar y pegar, respectivamente. Esto supone claramente una ventaja sobre FT2, cuyo sistema para este efecto es bastante menos intuitivo, además de tener algunos molestos "bugs".

Formatos soportados

Un punto extremadamente positivo de *MPT* es que es capaz de cargar un amplio abanico de formatos musicales. De esta manera podremos recuperar canciones que hayamos compuesto hace varios años, incluso en otros sistemas diferentes de los PCs, como Amiga o Atari ST. Los formatos soportados son los siguientes:



MPT nos permite aplicar efectos a la totalidad de la canción en tiempo de reproducción.

- *.669: módulos de UNIS 669, 669 Composer.
- *.AMF: módulos de Asylum / DSMI.
- *.AMS: módulos de Velvet Studio v1.x.
- *.DBM: módulos de DigiBooster Pro.
- *.DSM: módulos de DSIK music.
- *.FAR: módulos de Farandole Composer.
- *.IT: módulos de Impulse Tracker.
- *.MDL: módulos de DigiTracker 1.x.
- *.MED: módulos de OctaMed (sólo módulos MMD0/MMD1 válidos).
- *.MID: ficheros Midi (Los samples serán tomados de la librería MIDI).
- *.MOD: módulos de ProTracker (4-32 channels, 15/31 samples).
- *.MTM: módulos de MultiTracker.
- *.NST: módulos de NoiseTracker.
- *.OKT: módulos de Oktalyzer.
- *.S3M: módulos de ScreamTracker III.
- *.STM: módulos de ScreamTracker II.
- *.ULT: UltraTracker modules (buggy).
- *.WOW: módulos de Grave Composer.
- *.WAV: Carga ficherosWAV, separando canal izquierdo y derecho en dos samples.
- *.XM: módulos de FastTracker II.

Y los formatos comprimidos de los anteriores: (*.MDZ, *.S3Z, *.XMZ, *.ITZ, *.ZIP). La canción, para poder ser editada, será convertida a uno de estos formatos: MOD, S3M, XM ó IT, que son aquellos con los que MPT puede trabajar.

Un programa bien documentado

Disponemos de una ayuda hipertextual con índice, que nos será muy útil en caso de tener dudas de manejo del programa, pero también se nos da la posibilidad de acceder directamente a Internet desde el programa para acceder a una serie de lugares de interés cuyas direcciones IP tiene ya incluidas. Estos lugares son los siguientes: ModPlug Central, United Trackers, Trax in Space, The Tracker's Handbook y ModPlug Central Forums, donde podremos acceder a todo tipo de información relacionada con la composición musical por ordenador con secuenciadores. También se incluyen las direcciones de la página web del propio programador, Olivier Lapicque (http://www.ips.net/olivierl), y la página donde se puede obtener la última versión de *ModPlug Tracker* (http://www.modplug.com).

Estructura arbórea

Este secuenciador se sale bastante, a primera vista, de la estructura usual de los trackers, pero, si exploramos un poco las opciones del programa, pronto nos daremos cuenta que su manejo es muy similar al de FastTracker o Impulse Tracker, e incluso más sencillo que el de éstos ya que el entorno Windows conlleva siempre manejos más intuitivos.

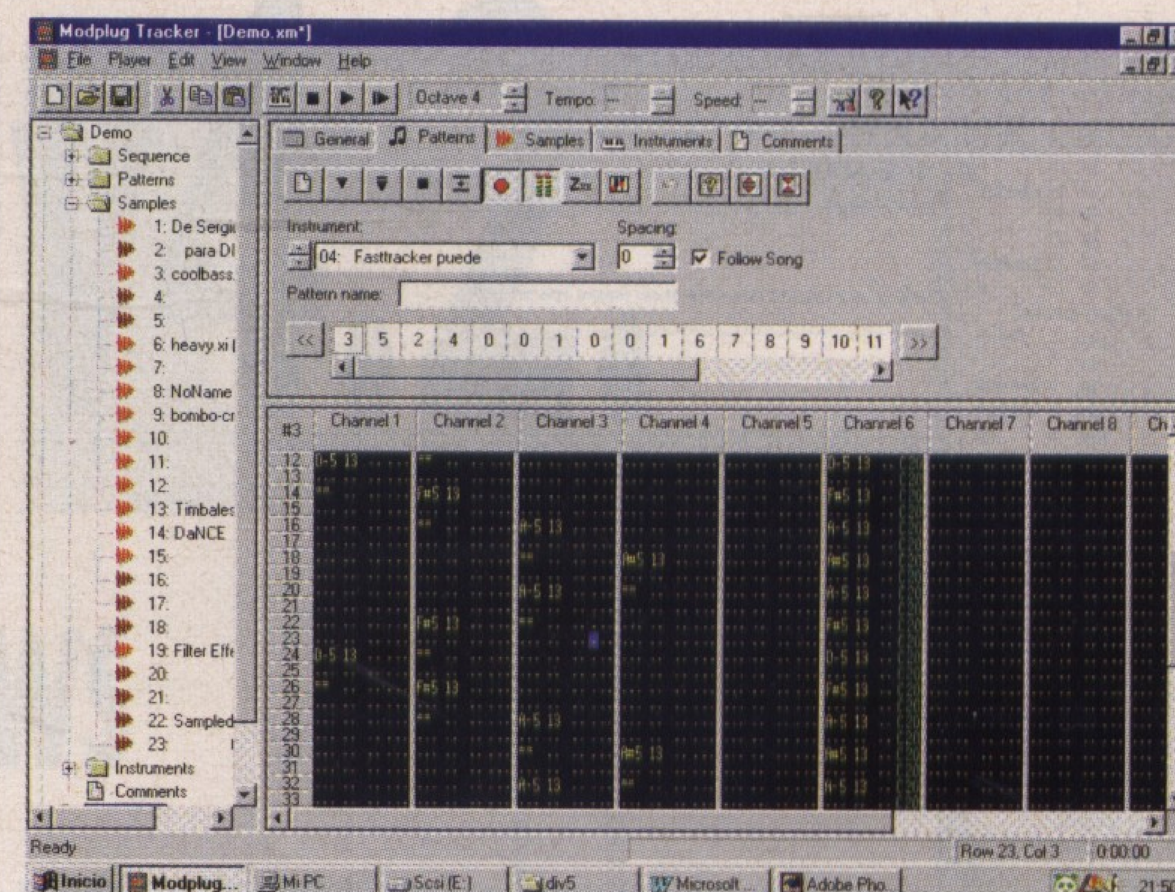
A la izquierda de la pantalla encontramos una estructura en forma de árbol, idéntica a la que utiliza el explorador de Windows. Cuando cargamos un fichero (Open) encontraremos los siguientes campos en esta estructura arbórea:

- **Nombre de la canción.** Si desplegamos este nodo aparecen 4 campos: Sequence (muestra el orden en el que están organizados los patterns que componen el módulo), Patterns (muestra una lista en orden de creación de todos los patterns del módulo), Samples (muestra una lista de los samples utilizados en la canción) e Instruments (instrumentos utilizados, ya que un instrumento puede englobar varios samples).
- **Midi library.** Desplegándolo se nos muestran los instrumentos MIDI que componen el banco MIDI que está seleccionado por defecto.
- **GM.DLS.** Es una lista de bancos de instrumentos MIDI (esto variará dependiendo de la tarjeta de sonido de la que dispongamos).
- **Instrument library.** Es una lista de paths en los cuales podremos encontrar instrumentos en nuestra máquina.

Pestaña "general"

En la mitad derecha de la pantalla se nos presentan una serie de pestañas con las cuales podremos acceder a los distintos editores del programa. Desde la pestaña "general" podremos controlar los siguientes parámetros:

- El título de nuestra creación.
- Initial settings (parámetros iniciales): el *tempo* (en negras por minuto), la velocidad de reproducción de la canción (*speed*), el volumen global del módulo (*global volume*) y la posición a partir de la cual deberá volver a empezar a reproducirse la canción cuando acabe de leer al último tracker (*restart position*).



El editor es muy similar al de FastTracker 2 o Impulse Tracker.

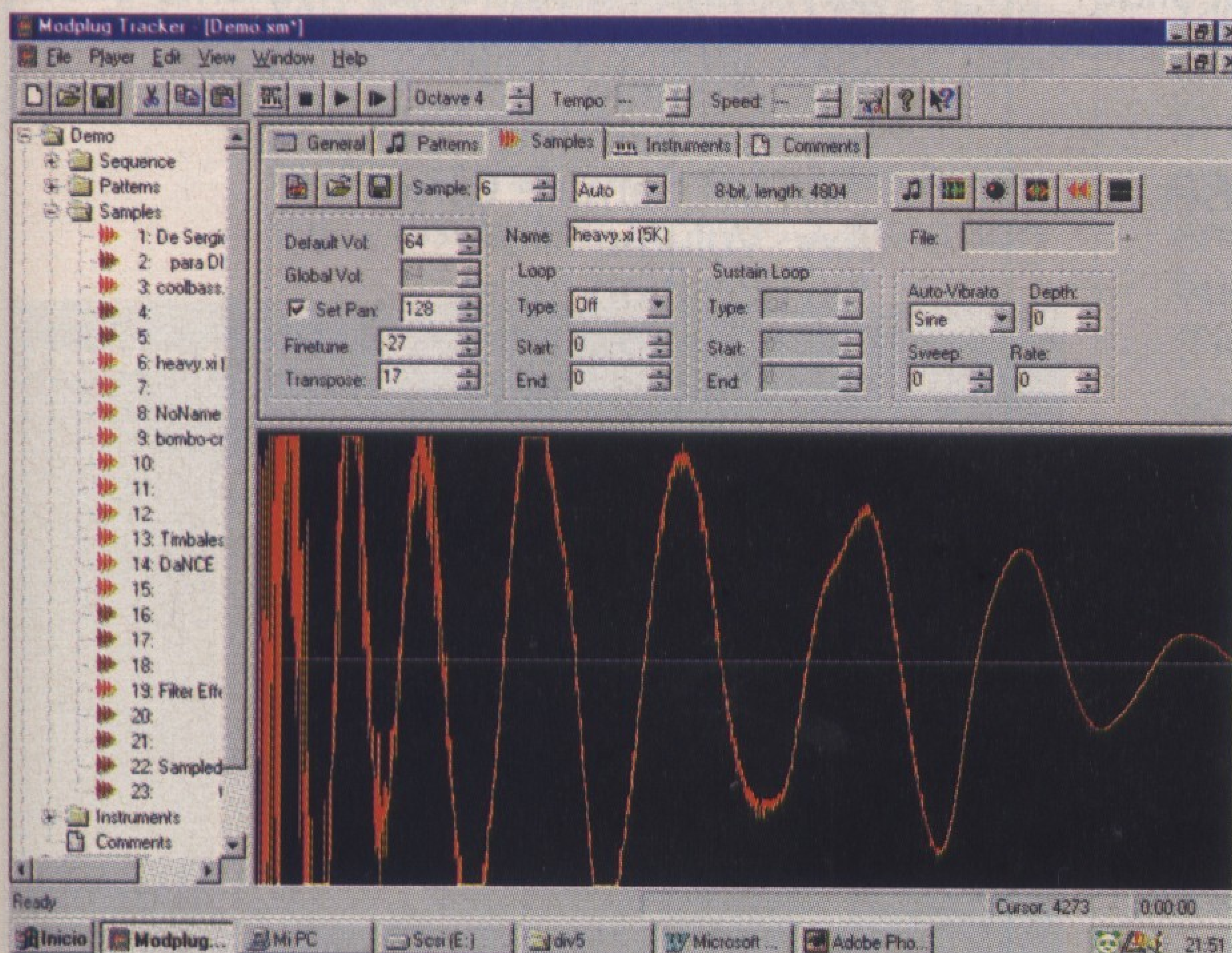
- Song type (tipo de la canción): sirve para hacer compatible nuestro módulo con Protracker (.MOD), Scream Tracker (.S3M), FastTracker (.XM) o Impulse Tracker (.IT). Para ello podremos seleccionar algunos puntos de compatibilidad: *Linear Frequency Slides, Fast Volume Slides, IT Old effects* e *IT Compatible Gxx.*
- Effects (efectos de ecualización): Esta opción nos permite aplicar algunos efectos de ecualización en tiempo real durante la reproducción. Estos efectos son *Bass Expansion* (realza los graves), *Reverb* (eco de sala), *Surround* (sonido envolvente), *Graphic ecualizer, AGC* y *Loop song* (reproducción cíclica de la canción).

Disponemos de un editor de patterns muy similar al de *FastTracker*

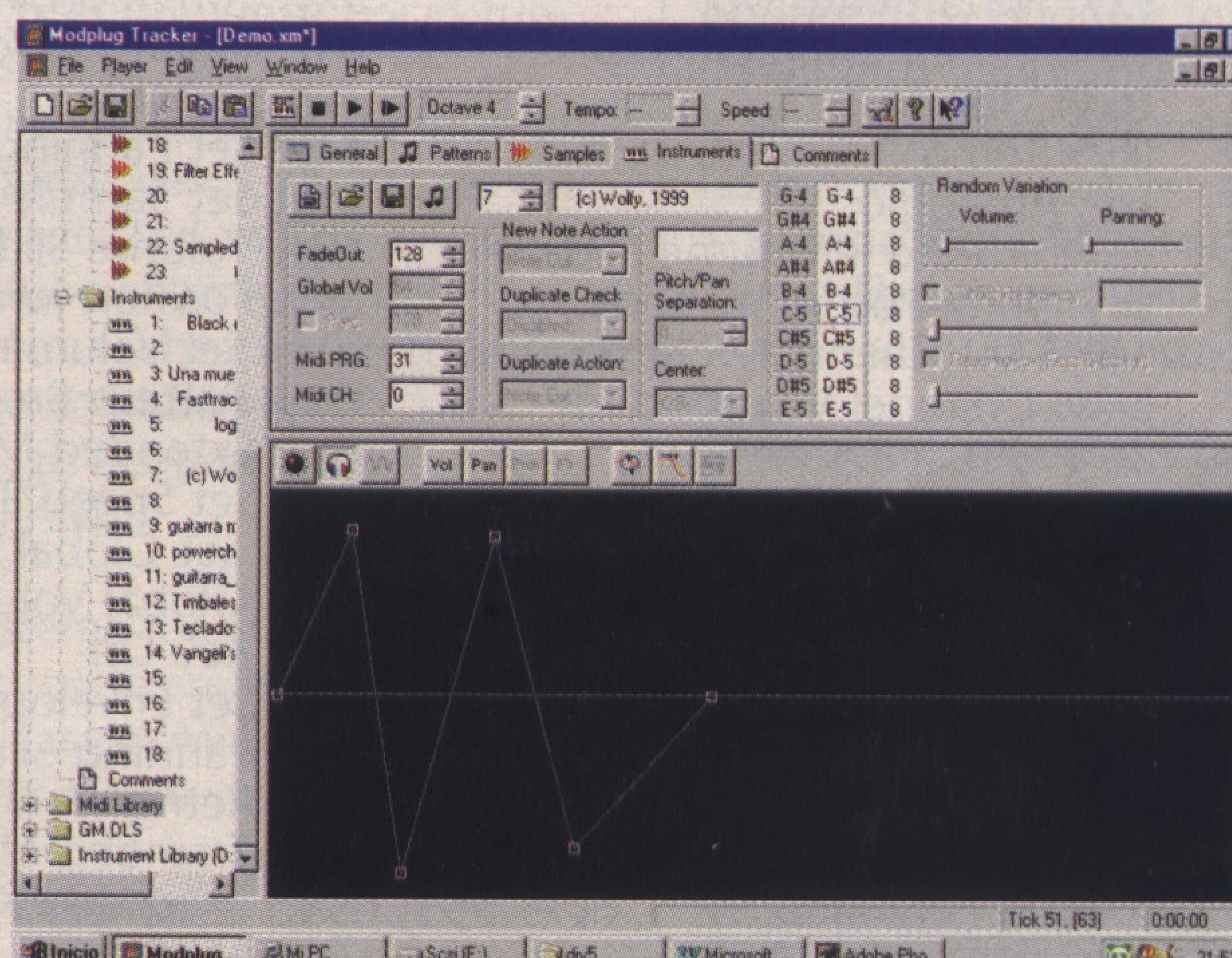
En la parte inferior de la pantalla y para cada canal podremos seleccionar el título del canal, su volumen, su panorámica y su habilitación/deshabilitación (mute).

Pestaña "patterns"

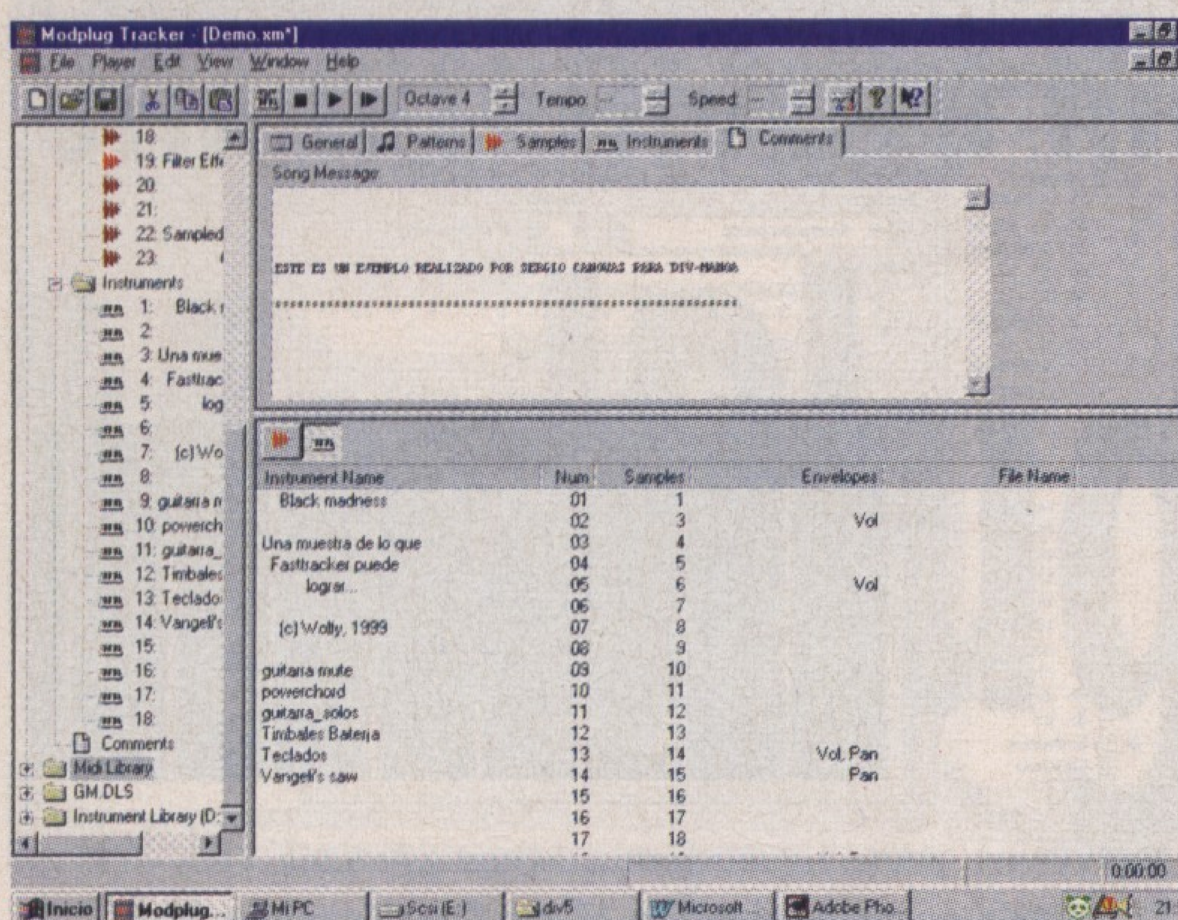
Es en esta opción donde encontramos el nexo de unión de *MPT* con



Disponemos de un editor de samples, pero no podremos digitalizar directamente con MPT.



El editor de instrumentos permite aplicar envolventes de volumen y panorámica, al igual que FT 2.



Ya no tendremos que escribir la dedicatoria en el lugar del nombre de los instrumentos.

el resto de secuenciadores, ya que es un editor de patterns extremadamente similar al de *FastTracker*. La única diferencia se da en la forma de especificar los efectos, que en el caso de *FastTracker*, sólo disponíamos de 3 dígitos para controlar el volumen, el vibrato, la panorámica, etc., y sin embargo en *MTP* se dispone de 6 dígitos, quedando 3 para el control de volumen y panorámica y otros 3 para el resto de los efectos. De esta manera, un ejemplo de línea de un canal podría ser el siguiente: C-5 01 V56 E22.

Un punto desfavorable: no podremos samplear directamente con *ModPlug Tracker*

En la parte superior aparece una barra de botones cuyas funciones (de izquierda a derecha) son las siguientes: Borrar el pattern actual, Play, Stop, Step (reproducción línea a línea), avanzar, Zxx (opción de macro para aglutinar varios efectos en uno para comodidad del usuario), editor de acordes (chord editor), pattern properties (número de líneas del pattern) y expandir/contraer pattern.

El editor de acordes, ya mencionado, es una opción muy interesante de *MPT*, recuperada de las versiones para ordenadores AMIGA del programa *ProTracker*. Permite escribir todo un acorde formado por diversas notas pulsando una única tecla, siempre que tengamos activada la opción *multichannel record*.

Pestaña "samples"

Haciendo "clic" en esta pestaña accedemos al editor de samples. Este es uno de los puntos más flojos del programa, porque *MPT* no puede samplear, sino tan sólo modificar una onda que carguemos desde disco. Si quisiéramos entonces especializarnos en la composición con *MPT*, no tendríamos más remedio que utilizar un programa adicional para digitalizar sonidos desde una fuente externa, como *Cool Edit 96*.

Con el editor de samples de *MPT* podremos modificar las opciones de visualización de la onda, copiar/pegar fragmentos, controlar el volumen y la nota del sonido con gran precisión (finetune), seleccionar las opciones de repetición de la onda, etc. También podremos ponerle vibrato a los samples, controlándolo de manera muy efectiva. En la barra de botones tenemos las opciones de reproducir/parar, amplificar la onda hasta el máximo posible (sin originar distorsión), amplificación libre (fade in, fade out), resample (para cambiar la frecuencia de muestreo de la onda en función de la longitud en bytes) y reverse (para invertir la onda).

El resto de las opciones

Las dos pestañas restantes son "Instruments", con la que accedemos a un editor de instrumentos muy similar al de *FastTracker*, mediante el cual podremos aplicar envolventes de volumen y panorámica a las ondas, además de algunas otras opciones, y la pestaña "Coments", con la que podremos añadir libre y extensamente comentarios diversos a nuestras creaciones musicales. Esta última opción es de agradecer, ya que en *FastTracker* y otros secuenciadores nos vemos obligados a buscar sitio en los lugares más insospechados para escribir estos comentarios, siendo el recurso más utilizado sobreescribir los nombres de los instrumentos.

En conclusión, *ModPlug Tracker* es un secuenciador bastante completo, que incorpora innovadoras y útiles herramientas. Especialmente recomendado para los "locos" de Windows.

Sergio Cánovas

Modos griegos (en Do M - La m)

IT (Impulse Tracker)	XM (Fast Tracker)	MOD (ProTracker)
Axy: Set Speed	0xy: Arpeggio	0xy: Arpeggio
Bxy: Position Jump	1xy: Portamento Up	1xy: Portamento Up
Cxy: Pattern Break	2xy: Portamento Down	2xy: Portamento Down
Dxy: Volume Slide	3xy: Tone-Portamento	3xy: Tone-Portamento
Exy: Portamento Down	4xy: Vibrato	4xy: Vibrato
Fxy: Portamento Up	5xy: Tone-Port + Vol Slide	5xy: Tone-Portamento + Volume Slide
Gxy: Tone-Portamento	6xy: Vibrato + Volume Slide	6xy: Vibrato + Volume Slide
Hxy: Vibrato	7xy: Tremolo	7xy: Tremolo
Ixy: Tremor	8xy: Set Panning	8xy: Set Panning
Jxy: Arpeggio	9xy: Set Sample Offset	9xy: Set Sample Offset
Kxy: Vibrato + Volume Slide	Axy: Volume Slide	Axy: Volume Slide
Lxy: Tone-Port + Vol Slide	Bxy: Position Jump	Bxy: Position Jump
Mxy: Set Channel Volume	Cxy: Set Volume	Cxy: Set Volume
Nxy: Channel Volume Slide	Dxy: Pattern Break	Dxy: Pattern Break
Oxy: Set Sample Offset	Exy: Extended MOD Commands	Exy: Extended MOD Commands
Pxy: Panning Slide	Fxy: Set Speed/Tempo	Fxy: Set Speed/Tempo
Qxy: Retrig	Gxy: Set Global Volume	
Rxy: Tremolo	Hxy: Global Volume Slide	
Sxy: Extended S3M Commands	Kxy: Key Off	
Txy: Set Tempo	Lxy: Set Envelope Position	
Uxy: Fine Vibrato	Pxy: Panning Slide	
Vxy: Set Global Volume	Rxy: Retrig	
Wxy: Global Volume Slide	Txy: Tremor	
Xxy: Set Panning	Xxy: Extended XM effects	
Yxy: Panbrello	Zxy: Macros	
Zxy: Macros		

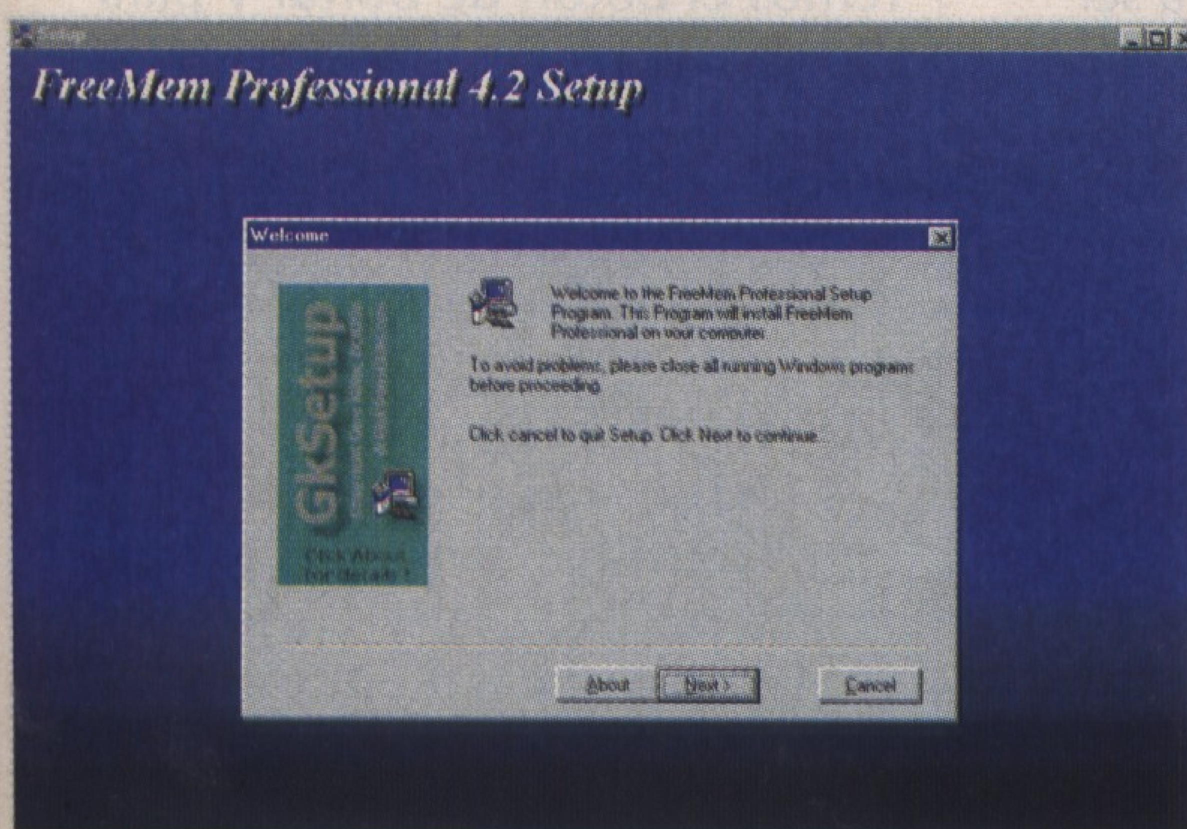
FreeMem Pro v4.2.

Liberador de memoria de Windows

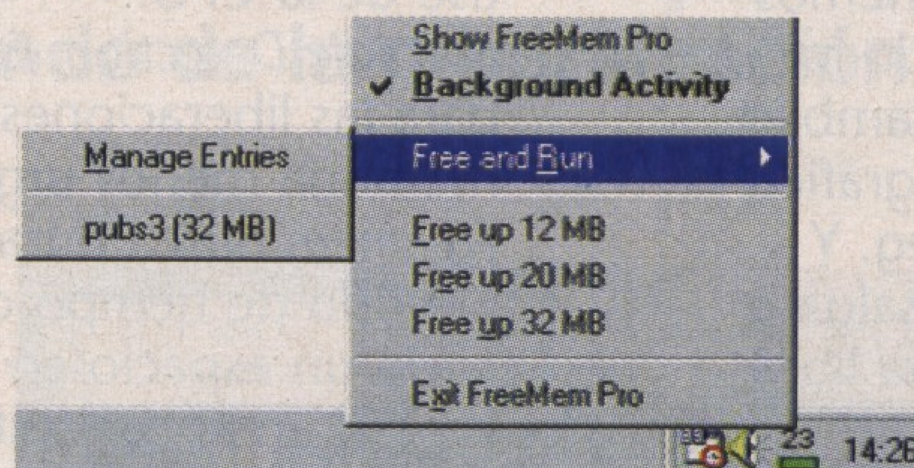
Al igual que en ediciones anteriores de esta revista, vamos a seguir nuestro recorrido por el shareware más actual y atractivo del mercado.
En este número veremos la utilidad de un programa de manejo de memoria RAM, en este caso el *FreeMem Pro v4.2* en su versión de 32 bits.

Empezaremos haciendo una breve introdución sobre el manejo de memoria de Windows. El proceso de manejo de memoria utilizado por Windows (95/98/NT/2K) hace posible que una aplicación que necesita más RAM de la que realmente se posee pueda ser ejecutada. Esto se consigue gracias a que este proceso de manejo de memoria utiliza una técnica llamada "swap" (escritura en disco de la memoria RAM no utilizada en ese momento por Windows) dejando así libre la memoria necesaria para otros recursos del programa. También hace posible ejecutar varios programas al mismo tiempo, cogiendo del disco los recursos necesarios en cada momento. Otra tarea de Windows es utilizar la memoria RAM como "cache" de ciertos ficheros que el sistema operativo cree que van a ser necesarios; con esta técnica se consigue agilizar muchas tareas u operaciones que de otra manera tendrían demasiados accesos a disco. Los accesos a disco son del orden de 1000 veces mas lentos que a la RAM, y la lectura de disco es 100 veces mas lenta.

Este programa acelera los procesos que necesitan mucha cantidad de memoria libre



Aspecto del programa de instalación.

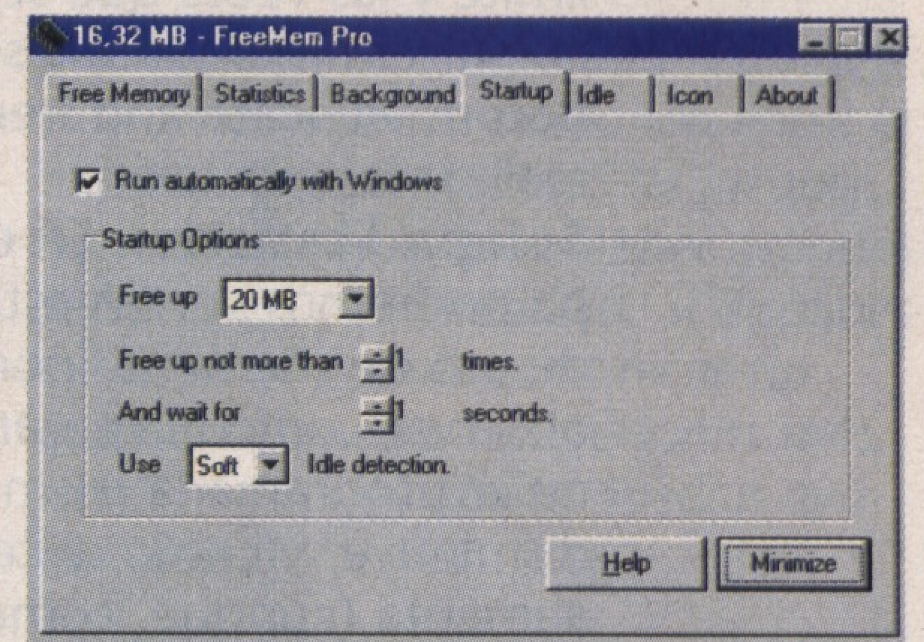


Aspecto del menú emergente del icono.

Pero este sistema tiene algunos fallos:

- Si un programa tiene algún error de programación, puede ser que parte de la memoria que se reservó para él no sea liberada al terminar el programa, con lo que Windows cree que sigue siendo utilizada (situación nada inusual).
- La utilización de grandes librerías de funciones por parte de los programas conlleva un gasto innecesario de memoria, ya que de esas librerías no van a ser utilizadas mas que unas pocas funciones.
- El cache de ficheros, aunque útil en algunos casos, puede ser que entorpezca, en vez de ayudar, el buen funcionamiento del PC. (Ficheros cacheados en memoria que no son accedidos y sin embargo ocupan hasta un 70% de la RAM).

El sistema más fácil para resolver estos problemas es ejecutar una aplicación muy grande que reserve la mayor parte de la RAM, causando que las partes no utilizadas sean escritas al disco. Esto es lo que hace *FreeMem Pro:* reserva la cantidad de RAM indicada, escribiendo en ella datos basura (necesario para que Windows se crea que realmente es utilizada) y acto seguido liberarla para que pueda ser utilizada por otras alicaciones.



Solapa "Startup": configuración al inicio de Windows.

Instalación y configuración

El programa viene en un archivo .ZIP; tendremos que descomprimirlo a una carpeta temporal y desde allí ejecutar "setup", con lo que nos aparecerá el programa de instalación. Elegiremos la carpeta de destino del programa y el grupo del menu Inicio que se creará. El resto del programa se instalará automáticamente.

Una vez instalado tenemos que configurarlo. Para arrancar el programa tendremos que ir al menú Inicio, Programas, FreeMem Professional y ejecutar FreeMem Professional. Nos aparecerá una ventana con varias solapas donde podremos configurar todos los parámetro del programa. Vamos a ir viendo solapa por solapa todas las funciones.

- **Solapa Free Memory**. Se utiliza para liberar memoria en un momento dado hasta un límite

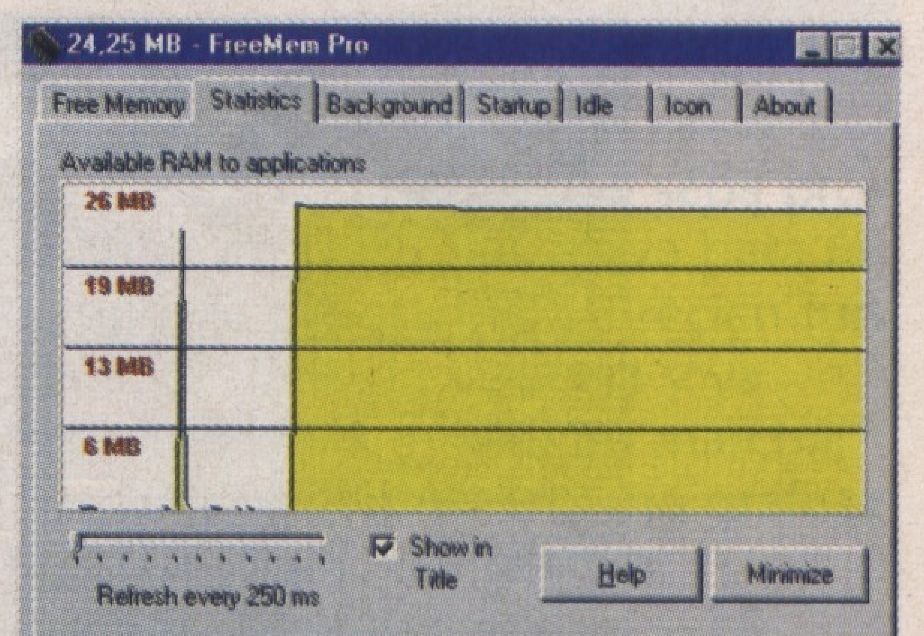
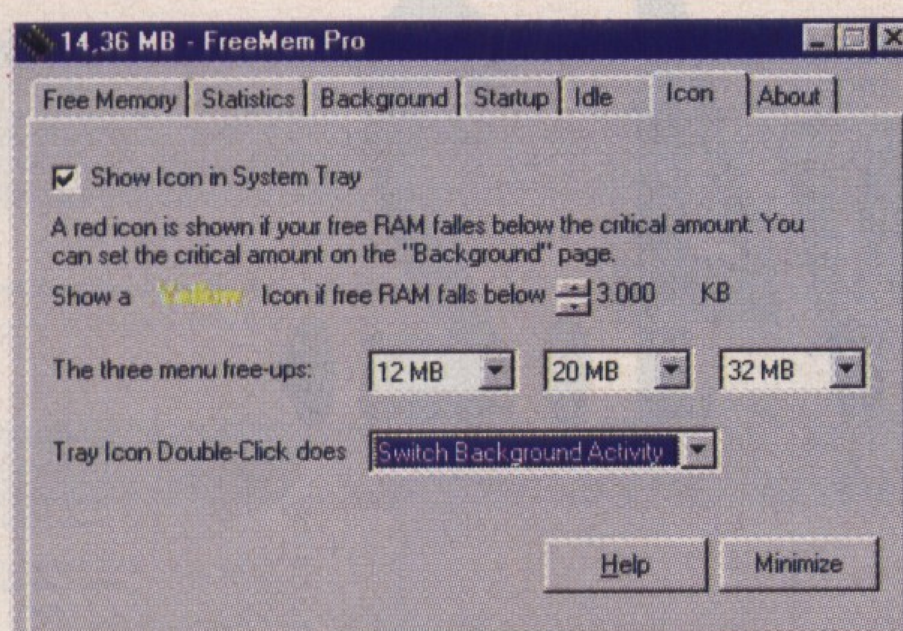
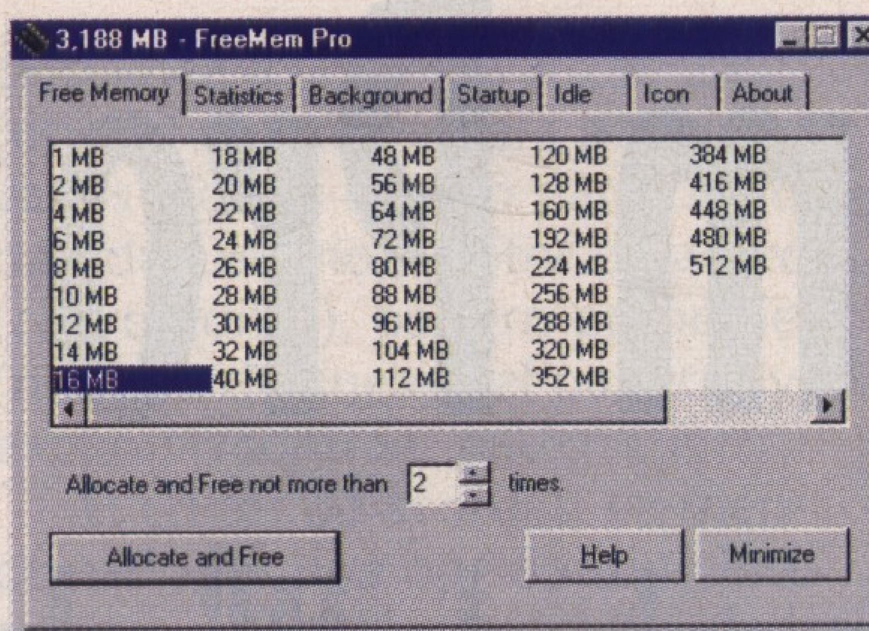


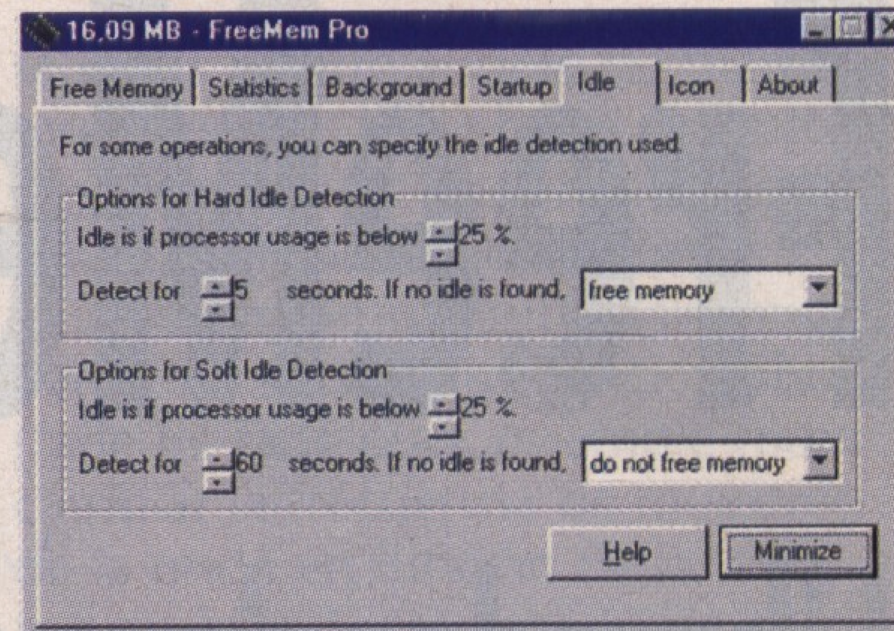
Gráfico con las estadísticas de la memoria.



Aspecto y propiedades del icono.



Liberación de memoria.



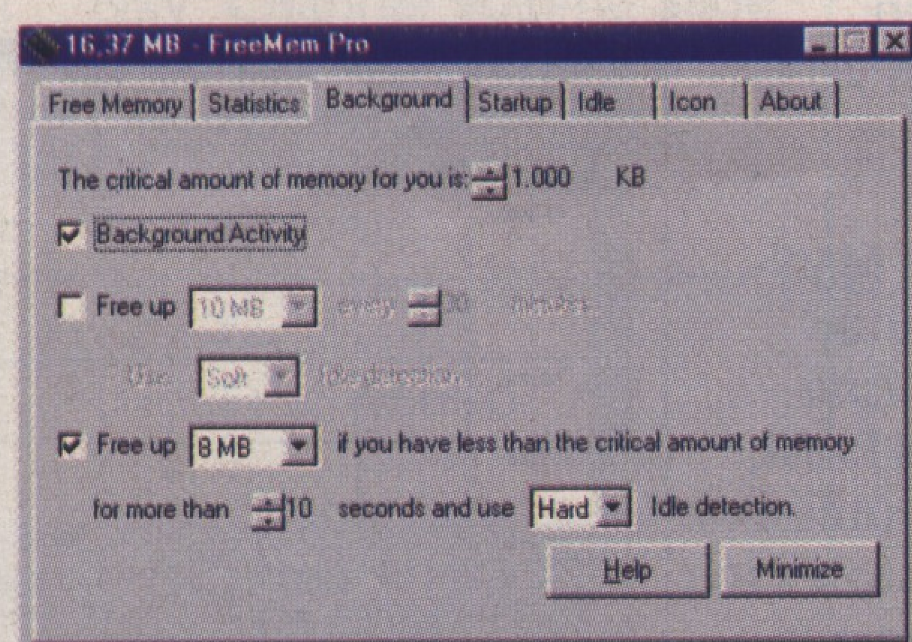
Distintos tipos de detección.

determinado. Se selecciona el límite y las veces que se intentará liberar esa memoria, y se pulsa el botón "Allocate and free" para realizar la operación.

- **Solapa Statistics**. En esta solapa podremos ver las estadísticas de uso de memoria; podremos observar como al cargar cualquier programa la memoria libre disminuye, y como al cerrarlo aumenta (aunque, como hemos dicho antes, quizás no todo lo que debiera). Podremos cambiar el tiempo de refresco del gráfico desde 250 ms hasta 60 seg. Y podremos optar por que salga o no la cantidad de memoria libre en el título del programa.
- **Solapa Background**. En esta solapa configuraremos cómo trabajará el programa en modo "background" (modo continuo). En un principio deberemos indicarle al programa la cantidad crítica de memoria (cuando se llegue a ella se pondrá en funcionamiento el sistema background). Una vez hecho esto, le indicaremos al programa si queremos actividad background o no señalando la casilla correspondiente, y en el caso de que sí la queramos, los pasos a seguir cuando se llega a la cantidad crítica de memoria o a un tiempo preestablecido.

Su configurabilidad es muy práctica para cualquier situación en que tengamos que liberar memoria

Se puede liberar memoria cada cierto tiempo hasta un cierto límite; para ello seleccionaremos la casilla adecuada y el intervalo de tiempo que habrá entre utilización y utilización y el modo de "Idle Detection".



Solapa para la configuración de la actividad "background".

También se podrá liberar memoria si se llega a la cantidad crítica de memoria; se intentará liberar durante cierto tiempo, y con cierto "Idle Detection".

Antes de proseguir deberemos explicar los distintos parámetros "Idle Detection"

- *No Idle Detection:* No existe ningún tipo de detección a la hora de liberar memoria del uso de la CPU.
- *Soft Idle Detection:* Es utilizada para las liberaciones de memoria menos urgentes. *Freemem* busca durante un periodo de tiempo determinado un espacio en el uso de la CPU en el cual esté libre y realiza su trabajo en ese momento; si no lo encontrara no realizaría la liberación de memoria.
- *Hard Idle Detection:* Esta opción está específicamente creada para liberar condiciones de RAM críticas. Buscará un espacio de tiempo en el cual la CPU esté libre (durante menos tiempo que en "Shoft Idle Detection") y si no lo encontrara, aún así liberaría memoria, interfiriendo en cualquier programa que esté en ejecución.

- **Solapa Startup:** En esta solapa se le indica al programa si se quiere que se ejecute al arrancar Windows o no, y si se quiere, cuanta memoria se intentará liberar, durante cuanto tiempo se intentará la liberación y el método de "Idle Detection" a utilizar.
- **Solapa Idle:** En esta solapa podremos definir qué van a ser "Soft Idle Detection" y "Hard Idle Detection". Podremos definir, para cada una de ellas, qué cantidad de CPU debe estar utilizada como máximo para que se activen y durante cuanto tiempo estarán esperando hasta esas condiciones, y lo que se hará si se sobrepasa el tiempo preestablecido.
- **Solapa Icon:** Aquí podremos configurar todo lo referente al icono del programa. Podremos decirle si al minimizar el programa queremos que sea un "Tray Icon" (los iconos pequeños que se encuentran al lado del reloj) o que se minimice normalmente. A qué cantidad de memoria libre queremos que cambie a color amarillo el icono; los menús de cantidad de memoria que será liberada en el menú desplegable del icono del programa (se accede a él haciendo clic con el botón derecho encima del icono); y lo que significa un doble clic sobre el icono (podremos optar por abrir la ventana del programa, activar/desactivar la función "background", y liberar ciertas cantidades de memoria en especial).

Función Free and Run

Una de las funciones que hacen más atractivo a este programa es la posibilidad de ejecutar cualquier programa habiendo liberado previamente una cantidad de memoria. Esta opción es muy buena para casos de programas que necesiten grandes cantidades de RAM y que no puedan ser ejecutados de otra manera.

Para acceder a esta función haremos clic con el botón derecho del ratón sobre el icono de *FreeMem Pro*. Se nos abrirá un menú con varias opciones entre las cuales se encuentra "Free and Run".

Dentro de "Manage Entries" podremos añadir, quitar o cambiar programas. Para añadir un programa deberemos indicar la cantidad de memoria a liberar, la ruta de acceso al programa en cuestión y un nombre que le queramos dar a esa entrada. Para borrarlo simplemente lo seleccionaremos y apretaremos el botón de borrar y para modificarlo apretaríamos el botón de modificar entrada.

Para terminar

Si queréis mas información sobre *FreeMem Professional*, los datos de Meikel Weber son:

E-Mail: *support@Meikel.com*

Web: *http://www.meikel.com/freemem*

Hasta el próximo número de la revista. Espero que os haya sido útil.

Javier Fernández

El correo del lector

Vuestro espacio

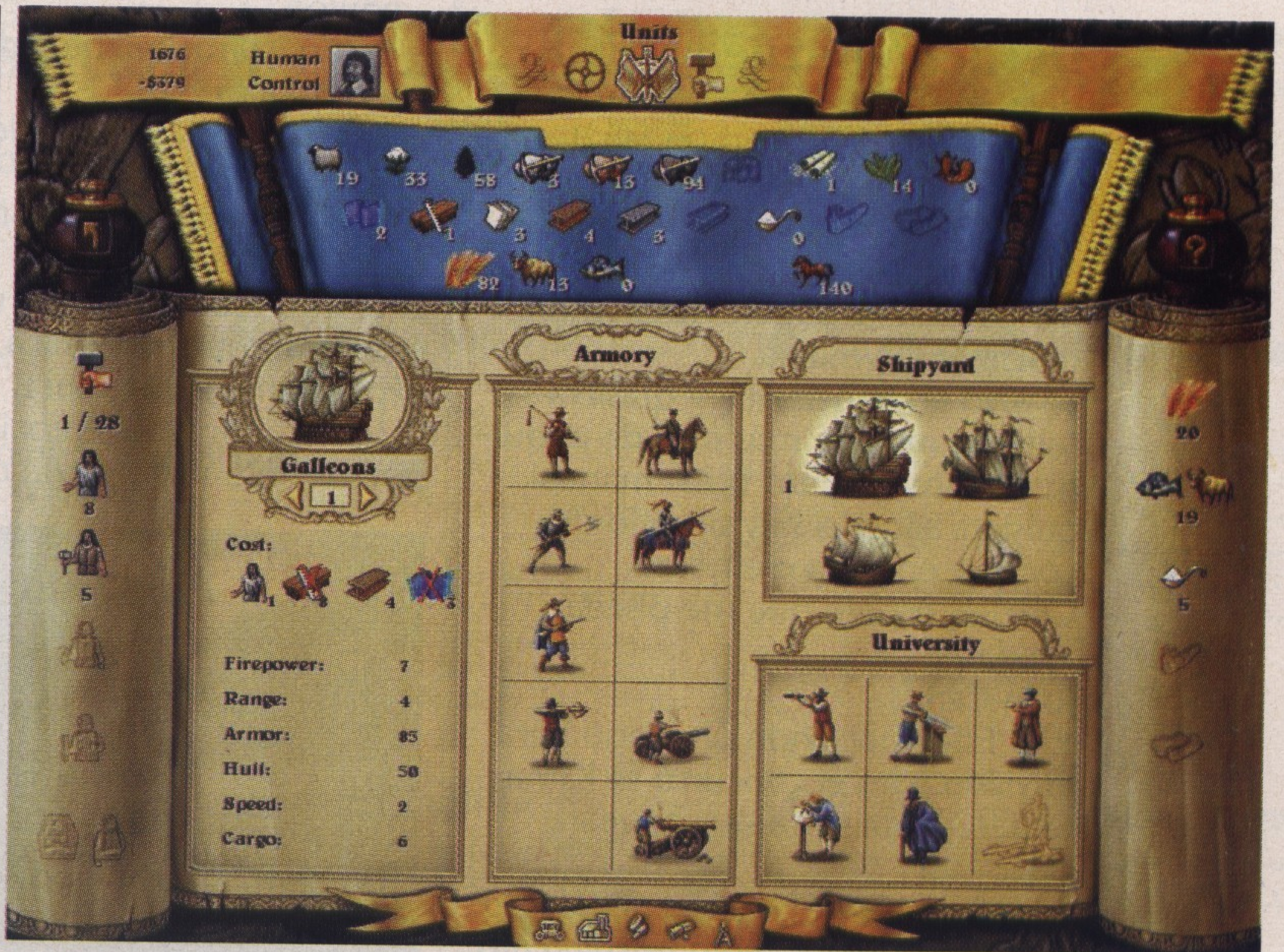
Todas las dudas que podáis tener acerca de la programación de juegos serán prontamente contestadas en este espacio, así que no lo dudéis, explicarnos vuestras dificultades y pronto dejarán de serlo. También recogemos, como es ya habitual, todas aquellas ofertas de trabajo que puedan interesar a todos los que estén inmersos en el mundo de la programación de videojuegos.

Hola, soy un chico de Bilbao de 17 años, y quiero ser desarrollador de videojuegos, preferiblemente diseñador, pero llevo 4 años buscando algo y nada. Quería una carrera universitaria, y pensaba en informática, pero no sé si me enseña lo que quiero, ya que es pura programación. Quería saber si sabéis algún sitio donde podría estudiar. También me ayudaría saber que han estudiado los de Pyro Studios, o Sega, o Nintendo, etc.

Bueno, la pregunta es difícil de contestar, pero aún así te daremos una respuesta para ver si te orientas un poco:

Hoy por hoy, una carrera de informática te da más conocimientos teóricos que prácticos, eso sin tener en cuenta las asignaturas de relleno para completar créditos, pero en cualquier caso siempre es una excelente elección cursar la carrera de informática. Aunque parezca a priori que algunas asignaturas no van a tener utilidad a la hora de desarrollar un videojuego, lo cierto es que ocurre todo lo contrario, desde física hasta matemáticas, pasando por bases de datos, todas tendrán su utilidad, aunque algunas más que otras. Debemos tener en cuenta que el desarrollo de un videojuego abarca una gran cantidad de aspectos, que no debemos menospreciar en ningún caso.

En cuanto a los conocimientos para desarrollar un juego, la mayoría de los que estamos en la actualidad programando videojuegos, para ser sincero, no se adquieren en una carrera universitaria. En este sentido, todos somos autodidactas, hemos adquirido nuestros conocimientos tras muchos libros en ingles, horas y horas de navegación en Internet, e infinitas pruebas con los ejemplos de Microsoft. También adquirimos nuestros conocimientos hablando entre la masa de programadores de juegos, normalmente no suele haber grandes secretos en este mundillo. Si dispones de Internet podrás comunicar tus inquietudes con otras personas más experimentadas que te puedan ayudar. Bye :)





Se busca gente joven

Soy Sacha, tengo 15 años y soy diseñador gráfico. Me interesaría contactar con gente que sepa programar en DIV, C++ o Java para un grupo de programación recién fundado. Somos de Valencia y ya tenemos un proyecto en marcha. También me ofrezco como grafista 2D si a alguien le interesa. Preferiblemente entre 14 y 16 años.

manuel.talens@uv.es







Desde Venezuela

Hola, me llamo Juan Carlos, me estoy convirtiendo en un fanático de DIV, he comprado su revista y me parece muy buena, yo siempre quise ser programador de Juegos y me parecía imposible hasta ahora, gracias a DIVmanía.

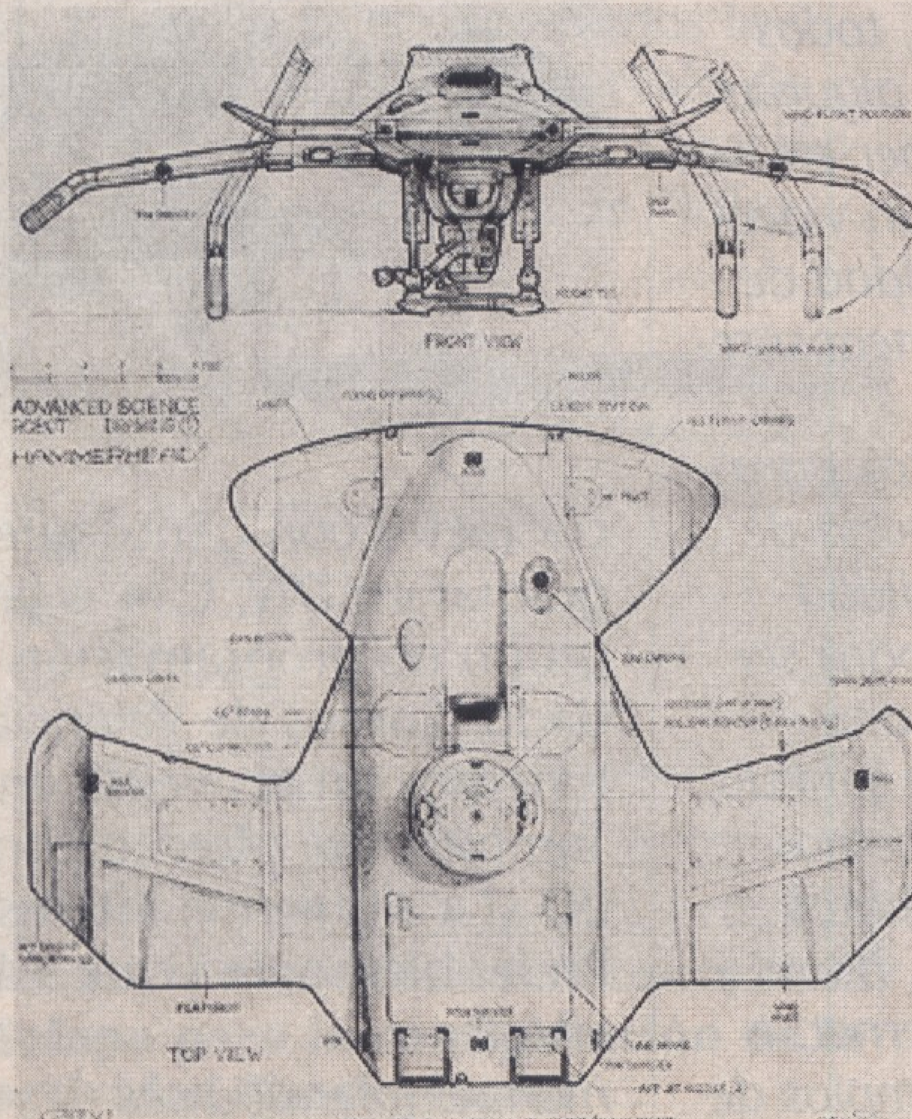
Yo vivo en Venezuela, y quisiera saber la manera más fácil de adquirir el software y un buen manual, quisiera saber si hay un sitio de compra en línea o algo así.

Gracias, espero que me puedan ayudar.

Juan Carlos

P.D. Estoy haciendo un juego para enviarlo para el concurso se llama SuperC, esperen a que lo vean.

Hola Juan Carlos, saludos desde esta redacción, ya nos contarás como has conseguido un ejemplar de DIVmanía en Venezuela, estamos bastante intrigados. Respecto a conseguir el programa DIV2 podrás hacerlo escribiendo y exponiendo tu petición en la siguiente dirección: suscripciones@prensatecnica.com. Esperamos ver tu juego pronto en nuestras manos. Hasta dentro de poco.



Se busca músico

Empresa líder en la edición y distribución de videojuegos, busca para sus estudios de producción:

SOUND DESIGNER

(Ref: SdLV28/11) para crear el entorno sonoro de un videojuego.

Requisitos:

- Titulados en música, escuelas de música, o similares.
- Preferentemente con experiencia informática musical y herramientas de edición de audio.
- Conocimientos del entorno PC (Windows 95 y/o NT).
- Gran creatividad artística.

Si además:

- Tienes facilidad para trabajar en equipo.
- Posees conocimientos de inglés y/o francés.
- Estas interesado/a por los videojuegos.
- Tienes: entre 22 y 30 años.

Si te interesa trabajar en una empresa joven, dinámica, en plena expansión e innovadora, envía tu CV indicando la referencia del puesto de trabajo y una carta de motivación. Todo ello por correo electrónico a:

Empleo@ubisoft.es,

por fax al número 93 590 62 53 o por correo normal a UBI STUDIOS,

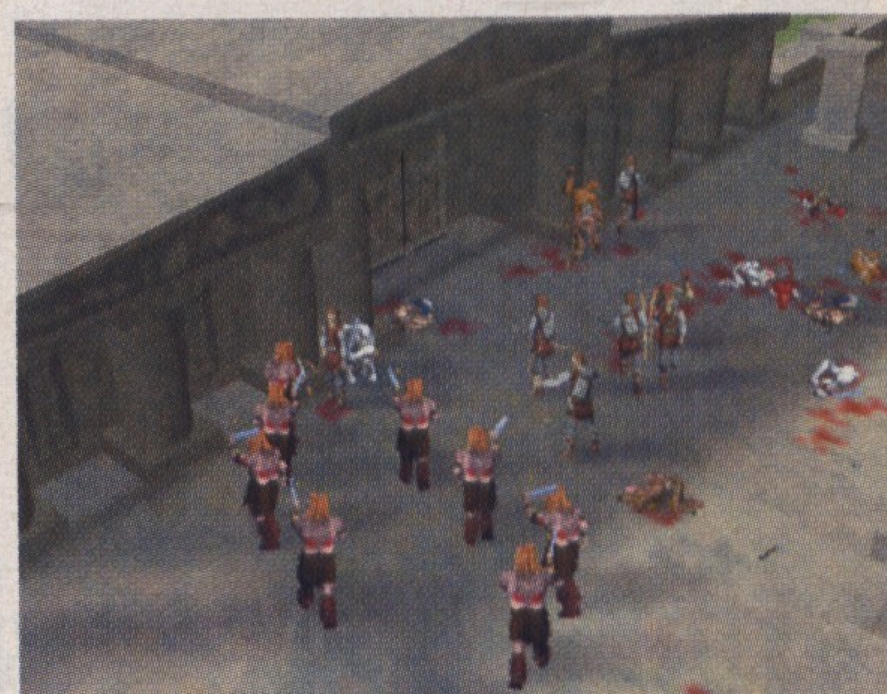
Crta. de Rubi nº72-74,

Edificio Horizon

08190 Sant Cugat del Vallès

(Barcelona).





Grafistas/músicos para juego de acción/rpg

Lince Games (miembro Stratos) busca a grafistas y músicos para el juego de rol Secret of *Lost Memory*.

El juego está siendo programado en Div 2, a 640x480 píxels, en perspectiva isométrica.

Está compuesto de dos partes, el modo historia, de un jugador; y el modo multijugador, en el que cada usuario maneja a un protagonista distinto, pudiendo ir de aventuras con los otros o por su cuenta. En este modo no habrá historia lineal, y cada uno irá haciendo su aventura.

Los escenarios han de ser ilustraciones en 2d, y los personajes, imágenes renderizadas en 2d.

También buscamos a programadores de apoyo, guionistas, etc. Todo tipo de ayuda viene bien.

Interesados llamad al 917333246 (solo tardes), o mandad un e-mail a *lincegames@mixmail.com*, o enviad una carta a:

LINCE GAMES

C/Cañaveral, 103, 1º C.

28029 Madrid.

Fe de errores

Estimados señores de Prens@ técnic@:

Rogamos disculpen las molestias que podamos ocasionarles al expresarles que hubo unos pequeños errores por nuestra parte al no incluir (por despiste y por las prisas para enviar el juego), al otro autor del proyecto del juego *Men In Green*; el coautor de dicho juego es Antonio Muñoz Calvillo (de ahí el nombre de AM, el otro player del juego, EJ soy yo, Emilio).

También nos gustaría comunicarles que el título es *MIG RAIDER, EIHN?*, aunque no sabemos si lo han puesto porque quizás parezca que *Men In Green* les parezca más adecuado (con más gancho).

Lo que pedimos en este e-mail es que se pueda buscar en la siguiente publicación un hueco para expresar nuestro error y que los lectores conozcan a los verdaderos creadores de *MIG RAIDER EIHN?*

Se despiden cordialmente unos lectores de "Divmania":

Emilio José Lopera Joyera & Antonio Muñoz Calvillo.

PROGRAMAR JUEGOS ES COSA DE NIÑOS SI TE SUSCRIBES

a Div Manía



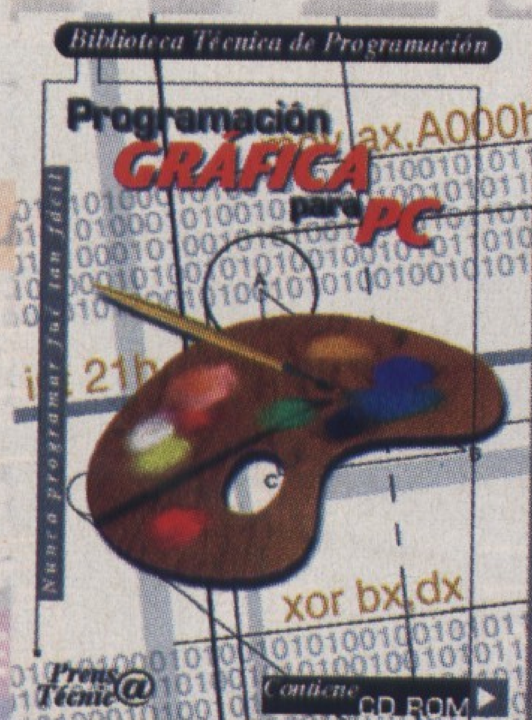
Si deseas estar en la vanguardia del mundo de la informática, suscribirse a DIV MANÍA es un primer paso acertado porque...

- *Es la única revista escrita por y para los programadores de videojuegos. Nuestra redacción está compuesta por veteranos desarrolladores, expertos del entorno DIV, grafistas y muchos otros profesionales del software de entretenimiento que dan lo mejor de sí a los lectores.*
- *Te ofrece lo último en el delicado campo de la programación de videojuegos, con los títulos que se encuentran en proceso y los productos recién salidos del horno o de los PCs.*
- *Para seguir avanzando hay que saber echar la vista atrás y a la vez no olvidarse del futuro, y **Div Manía** empieza un nuevo camino.*
- *Nuestra revista presenta un look muy cercano a sus lectores, salido de las inquietudes de todos vosotros.*
- *Nunca nadie te ha ofrecido tanto por tan poco; nunca has tenido tan cerca la oportunidad de estar al día de lo último en programación por el mínimo esfuerzo de acercarte al quiosco o enviar nuestro cupón y recibir la revista en casa puntualmente cada dos meses.*
- *En el interior del CD-Rom encontrarás los elementos con los que todos los programadores sueñan.*
- *Somos como tú y conocemos, más o menos, qué se esconde dentro de tu cabeza. Y si no lo conocemos aún, lo aprenderemos gracias a ti.*
- *Ofrecemos las más diversas sorpresas, las más interesantes ofertas, para que no te olvides de que la programación siempre está viva.*

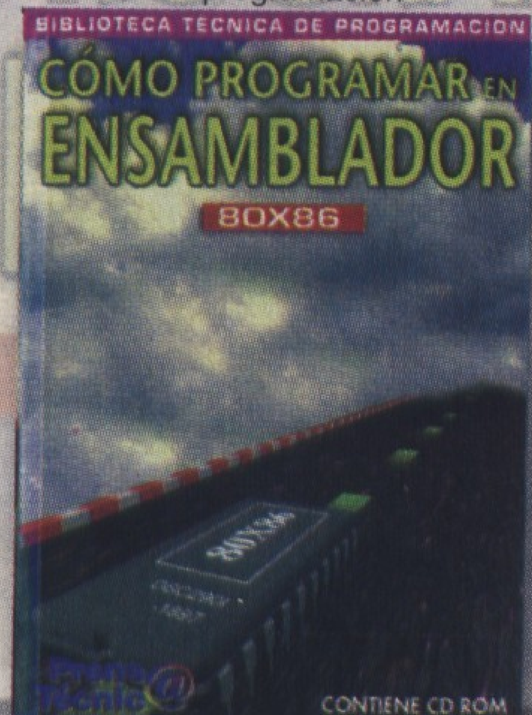
demás, el suscriptor tiene derecho la siguiente oferta:

Con un año de suscripción (seis números) regalamos un producto a elegir entre:
"Programación Gráfica para PC"
"Cómo programar en Ensamblador"

Con dos años de suscripción (doce números) regalamos DIV 2



PROGRAMACIÓN GRÁFICA PARA PC
Libro programación



CÓMO PROGRAMAR EN ENSAMBLADOR
Libro programación



DIV II
Creación de juegos

Solicite su ejemplar enviando este cupón por correo, por fax: 91 304.17.97 o llamando al teléfono 91 304.06.22 de 9:00 a 19:00 h.

CUPÓN DE SUSCRIPCIÓN ANUAL A DIV MANÍA

Deseo suscribirme a la revista **DIV MANÍA** acogiéndome a la siguiente modalidad:

☐ Suscripción: 1 año (6 números) por sólo 5.970 ptas. ☐ Correo certificado 1 año: 1.500 ptas. adicionales
☐ Suscripción: 2 año (12 números) por sólo 11.940 ptas. ☐ Correo certificado 2 años: 3.000 ptas. adicionale

Desde el número ☐

Además recibiré gratis:

☐ Por 1 año de suscripción: **uno** de los siguientes productos:
☐ Por 2 años de suscripción: **DIV II**

☐ Programación Gráfica para PC ☐ Cómo programar en Ensamblador

Nombre y apellidos ....
Domicilio ....
Población ....
C.P.... .... Provincia ....
Telf. .... Profesión ....
DNI/NIF: ....

FORMA DE PAGO:

☐ Con cargo a mi tarjeta VISA nº__________________ más gastos de envío
Fecha de caducidad de la tarjeta ....
Nombre del titular, si es distinto ....
☐ Domiciliación bancaria, más gastos de envío
Población ....
Ruego a Vd. que se sirva cargar en mi:

☐ cuenta corriente
☐ libreta de ahorro número

ENTIDAD	OFICINA	DC	Nº CUENTA

el recibo que será presentado por PRENSA TÉCNICA S.L. como pago de mi suscripción a la revista **DIV MANÍA** más gastos de envío.

FIRMA:

☐ Contrarreembolso del importe más gastos de envío.
☐ Cheque a nombre de PRENSA TÉCNICA, que adjunto más gastos de envío.
☐ Giro Postal (adjunto fotocopia del resguardo) más gastos de envío.

Prensa Técnica de libros y publicaciones

C/ Alfonso Gómez, nº 42 Nave 1-1-2. 28037 Madrid. Tfno: 91 304 06 22. Fax: 91 304 17 97
e-mail: maspc@prensatecnica.com. http://www.prensatecnica.com

Prensa Técnica se reserva el derecho de cambiar cualquiera de los regalos por otros de igual valor, en caso de agotar existencias.

Contenido CD-Rom



En este número seis, el CD-Rom que acompaña a la revista está a rebosar de contenido, además del que seguro os tendrá enganchados al ordenador durante unas cuantas horas, pasamos a resumiros lo que podréis encontrar.

JUEGOS GANADORES

Nuestro concurso sigue contra viento y marea. Estos son los ganadores de este número:

Tokencraft

Un homenaje a *Totenkai* y a *Warcraft*, que convierte a este juego en un arcade, protagonizado por el personaje principal del primero, que deberá acabar con los excombatientes orcos que escaparon con vida de las pantallas del segundo. Una buena idea que ha merecido el primer premio.

Nowadays

Un juego que evoca a las antiguas máquinas recreativas del género arcade. Os ponemos en situación: hay que manejar un avión que encontrará los cielos plagados de enemigos. Menos mal que nuestra pequeña nave es capaz de disparar sin cesar y limpiar la atmósfera de todos los enemigos que encontremos.

Alex Exoddus

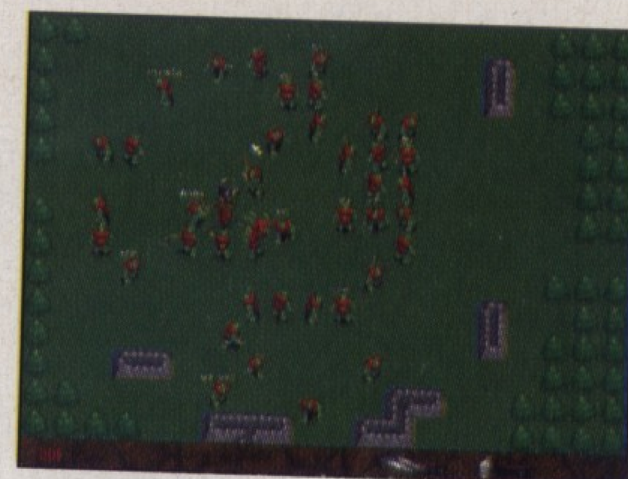
Un juego de plataformas muy adictivo en el que deberemos manejar a un protagonista (parece que un forofo del Atlético de Madrid) a través de un mundo lleno de enemigos. Varios niveles y grandes dosis de dificultad para acabarlos. Este es el juego que se ha alzado con el tercer premio.

REVISTAS ELECTRÓNICAS

Este mes ampliamos nuestro espectro. Además de la revista Divnet que ya conocéis, os ofrecemos otra interesantísima revista electrónica sobre Div: Vital-web.

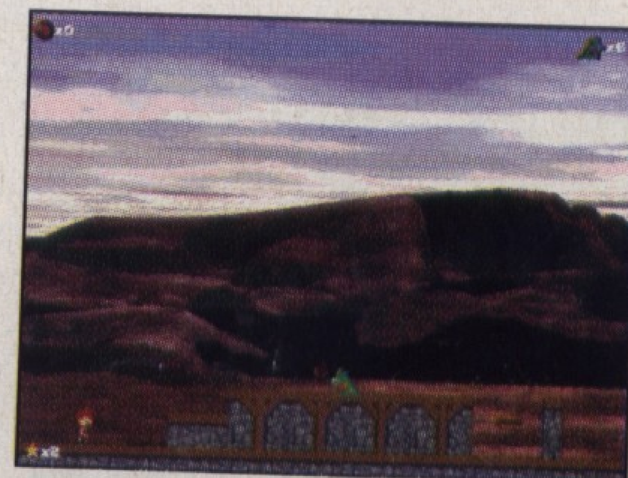
FONDOS DE ESCRITORIO

Sacha Talens, un lector de nuestra revista, nos ha ofrecido unos fondos de escritorio para ofrecerlos en el CD y, por supuesto que los hemos aceptado.



CURSO DE JUEGOS

Todos los complementos que necesitas para seguir todos las lecciones de nuestros cursos. Casi todas las secciones llevan códigos que por su extensión no es posible meter en la revista. En el Cd-Rom los encontrarás enteros. Además hay pequeños programas que seguro os serán de utilidad.



ESPECIAL JUEGOS

Todos sabemos que nuestro concurso de programación en Div, más allá del premio en metálico, es una oportunidad de dar a conocer las creaciones de todos los Divmaniacos. Todos y cada uno de los juegos que nos llegan merecen la pena, aunque sólo sea porque se nota que rebosan ilusión. Muchos lectores nos habían hecho llegar un mensaje muy claro: publicadlos, aunque no los premiéis. Pues

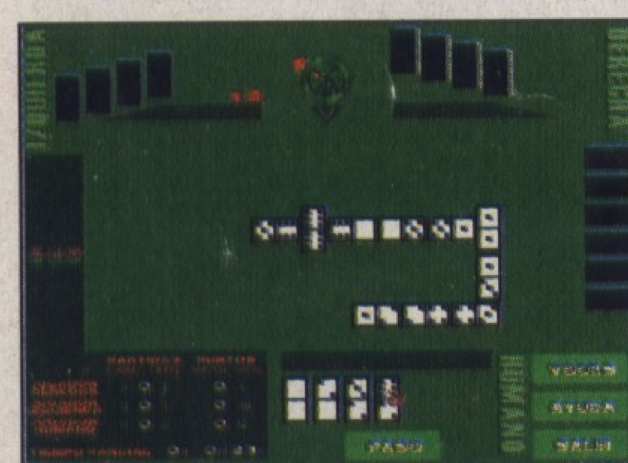
















bien, ahí va. Lo cierto es que tienen toda la razón del mundo, de modo que hemos decidido dedicar nuestro Cd a publicar los juegos que se han presentado al concurso.

Una pequeña aclaración: no están todos los que son pero sí que son todos los que están. Es posible que algunos de los juegos que hemos recibido en toda nuestra andadura no aparezcan en el Cd; la razón es sencilla: a lo largo de todo este tiempo hemos sufrido muchos problemas que pueden haber ocasionado la pérdida de algunos de ellos. Por eso pedimos perdón a todos nuestros lectores. Del mismo modo pedimos disculpas si alguno de los juegos no se ejecuta correctamente. En cualquier caso, estamos seguros de que merece la pena echarles un vistazo a todos y cada uno de ellos.

Aparte de que la gran mayoría son muy jugables y os divertirá probarlos, sus códigos fuente os ayudarán a aprender de los aciertos y errores de otros, y puede que os den la pista para solucionar ese problema de programación que os ha dejado atascados en algún punto de vuestros propios trabajos. Que los disfrutéis.

Accelerator: Jordi Bergós Massagué.

Adamchím: Adám Camacho Martín.

Arena: David Yuste.

Aselo: Francisco Blázquez Munuera.

B-Sides: Paulo Jorge Gomes Feitor.

Billar: Santiago Jiménez.

Bloody´s War 99: Julio Gorgé Frochoso.

Cd Voyage: Jordi Bergós Massagué.

Cinquillo: Antonio Marchal.

Comando Pelota: David Ferriz.

Dardos: Javier Alcubierre

Devillish: David Ferriz Blánquez.

Disco Fighter: E. González, Cristóbal García y C. Casás.

Ejdomino: Emilio José Jiménez Jiménez.

Exit: Josep Portella Florit.

Exploss: Ismael Fernández.

Exterminador de Pitufos: D. Fernández y F. Cuadrado.

Fumigasión Galástica: Álex Hernández Ortega.

Get3!: José Luis Parra Lozano.

Kubox: José Antonio Mango.

Manga Paradise: Carlos Pastor.

Memorion: Antonio Puig.

Mig Raider Eihn: Emilio José Lopera Joyera y Antonio Muñoz Calvillo.

Mini Dragón Ball Z: Oscar Rincón Jiménez.

Misión Marte: David Yuste Romero.

Mr. Bones: Víctor Reyes, Carlos López y Miguel López.

Navidad: Daniel García Alonso.

Neowing: Lui Bega.

Nuevo Milenio: Jacob Almagro García.

Numbers: Félix Martínez Vera.

Pa-Pú: Emilio Galindo y Jacob Almagro.

Parchís: Gonzalo de Lucio de la Iglesia.

Pingüinoid: David Ferriz Blanquer.

Pompas: Ángel Almodovar Moreno.

QA: Roberto Selva.

Rapid: Jordi Bergós Massagué.

Resident Leming: Enrique Medina Gremaldos.

Reyes: Javier Alcubierre Villasol.

Run or Die: Tomás Arribas Simón.

Saimaza: Juan Armada.

Sam & Tom: Jose Antonio López López.

Shootsex: Cristian Duro.

Space Opera: Ferrán Clavero Estrada.

Super 15: José María Cortés López.

Super Action Rajao 2: Juan Manuel Cebolla Carbonell.

Super Yohsy: Enrique Medina Gremaldos.

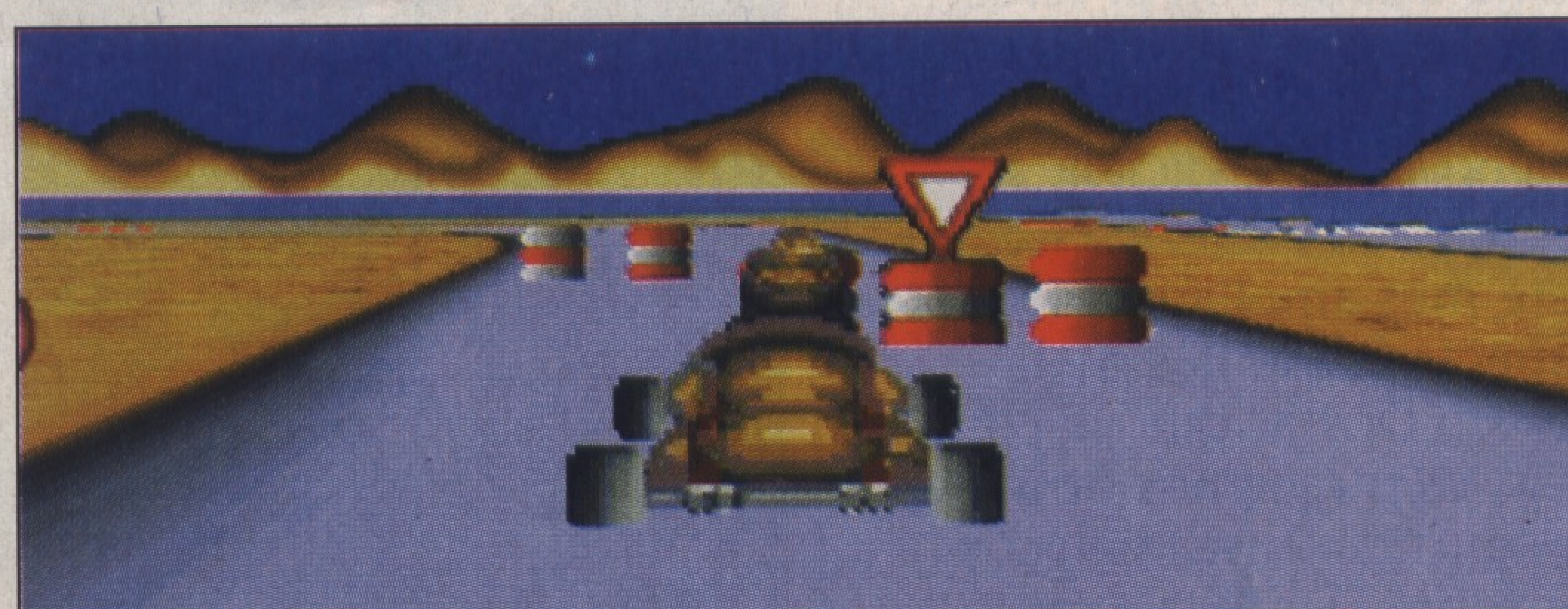
Tcow: Luis Ignacio Díaz del Dedo.

The Bombs: Pere Estany Barrera.

Tiro al Plato: Pedro Miguel Jiménez.

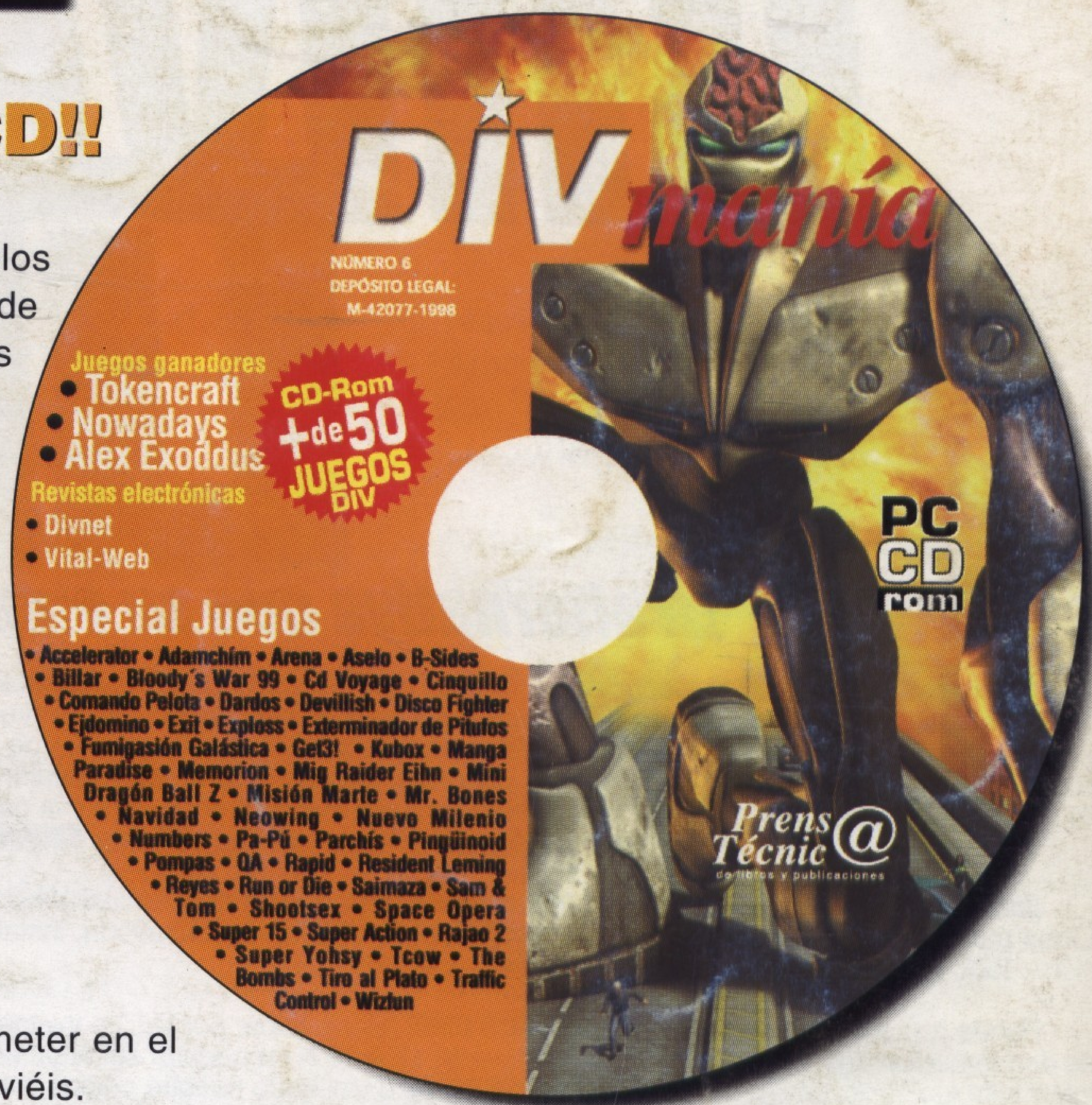
Traffic Control: Javi Guerrero Castroviejo.

Wizfun: D. Fernández Montañés.



CONTENIDO DEL CD ROM

¡¡MÁS DE 50 JUEGOS EN ESTE CD!!



Ponemos a disposición de los lectores de DIVmanía el fruto de los desvelos de la pléyade de programadores que han optado al premio del concurso de programación de esta revista. Para que aprendáis de sus aciertos, de sus errores y, sobre todo, para que os divirtáis con ellos, algunos son excelentes; otros simplemente divertidos; todos merecen la pena.

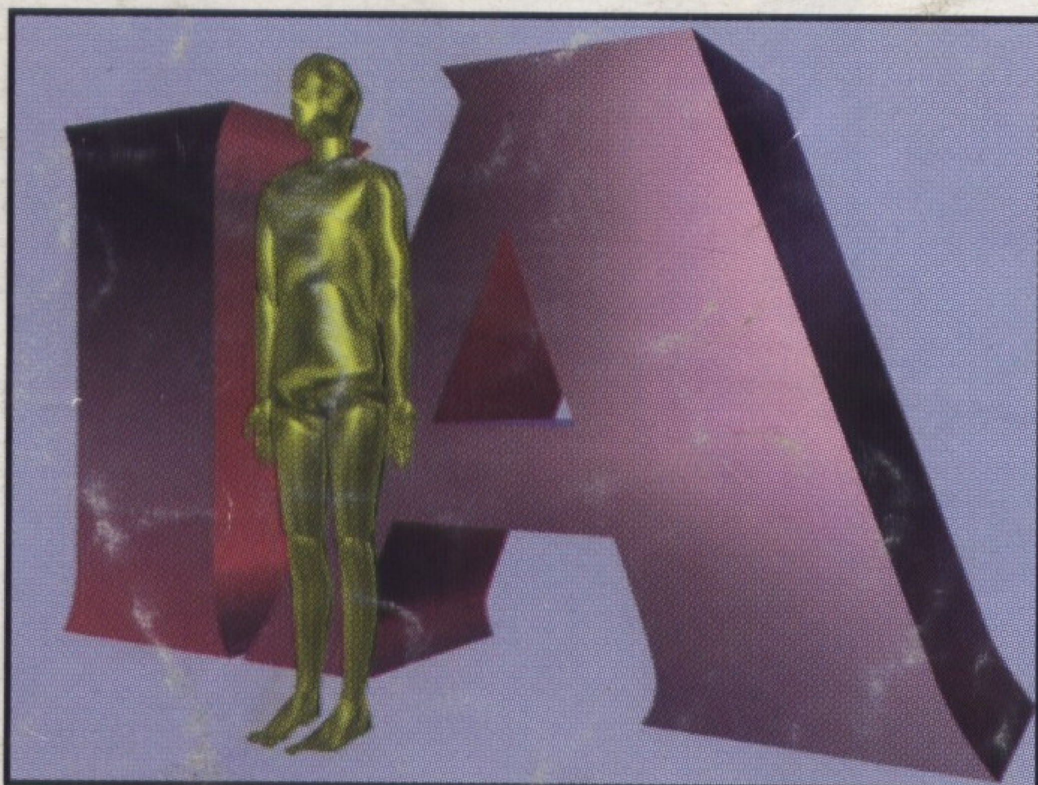
LOS TRES GANADORES
Mención aparte merecen los tres juegos ganadores de este mes, también disponibles en este CD-Rom.

REVISTAS ELECTRÓNICAS
El segundo número de Divnet y la presentación de Vitalweb al completo, dos estupendas revistas dedicadas a DIV y a la programación de juegos. Para todos aquellos que no dispongan de conexión a Internet.

FONDOS DE ESCRITORIO
Un lector de esta revista nos ha ofrecido unos fondos de escritorio para meter en el CD. Por supuesto que sí, si hay sitio meteremos todo aquello que nos enviéis.

CURSOS DE JUEGOS
Todos los complementos que necesitas para seguir todos las lecciones de nuestros cursos. Casi todas las secciones llevan códigos que por su extensión no es posible meter en la revista. En el Cd-Rom los encontrarás enteros. Además hay pequeños programas que seguro os serán de utilidad.

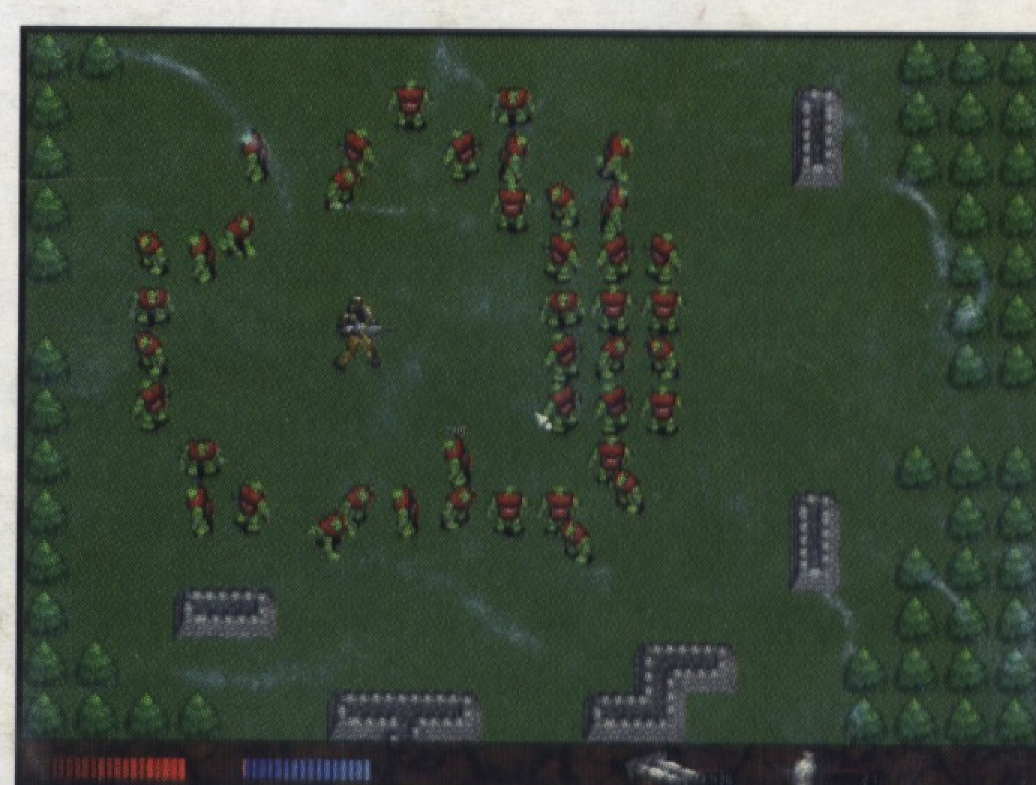
REPORTAJE. Lo que debes saber sobre la Inteligencia Artificial.



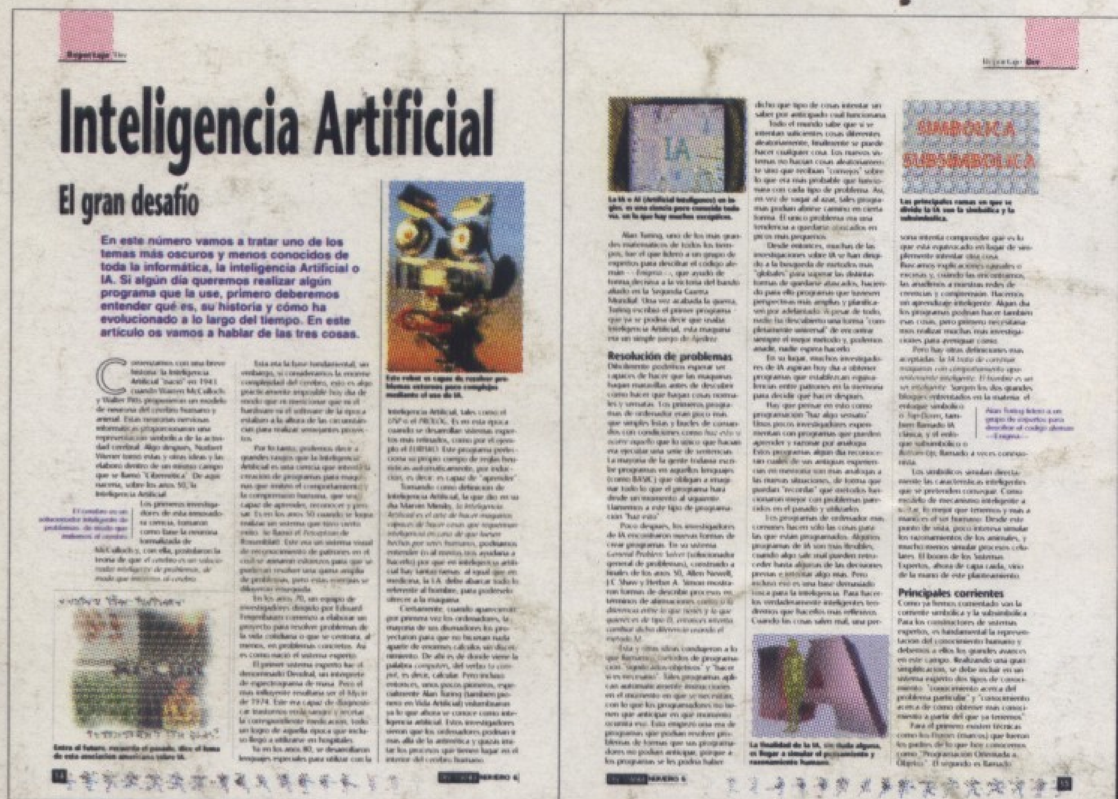
FONDOS. Cedidos por uno de nuestros lectores esperando que os sean útiles.



CONCURSO DE JUEGOS. Os presentamos a los ganadores de este mes y el código de sus juegos.



DIVmanía CON EL MEJOR CONTENIDO


Inteligencia Artificial
El gran desafío


Cadenas de texto
Tipos de datos "string"


Havas
Gigante con pies firmes

ACTUAL **EXHAUSTIVO** **DIDÁCTICO**

Y MUCHO MÁS...